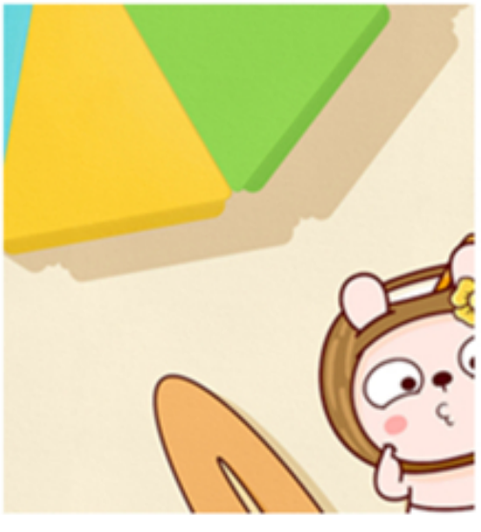
最新IT书籍PDF
最新IT书籍PDF
最新IT书籍PDF
最新IT书籍PDF



版权相关注意事项：
1、书籍版权归著者和出版社所有
2、本PDF来自于各个广泛的信息平台，经过整理而成
3、本PDF仅限用于非商业用途或者个人交流研究学习使用
4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负
5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
7、请于下载PDF后24小时内研究使用并删掉本PDF

版权相关注意事项：
1、书籍版权归著者和出版社所有
2、本PDF来自于各个广泛的信息平台，经过整理而成
3、本PDF仅限用于非商业用途或者个人交流研究学习使用
4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负
5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
7、请于下载PDF后24小时内研究使用并删掉本PDF

非卖品，仅供非商业用途或交流学习使用
开课吧 kaikeba.com | 慧科研究院 Huikedu Research | 高校邦 gaoxiaobang.com
严禁网络传播本PDF，违者责任自负！
中国计算机学会教育工作委员会委员 管刚
慧科集团技术副总裁 李嘉
版权所有，严禁传播，违者自负法律责任！

微信小程序实战入门

非卖品，仅供非商业用途或交流学习使用
严禁网络传播本PDF，违者责任自负！

（第2版）

版权所有，严禁传播，违者自负法律责任！
非卖品，仅供非商业用途或交流学习使用
严禁网络传播本PDF，违者责任自负！
版权所有，严禁传播，违者自负法律责任！

- 内含完整实例解析
- 第2版内容更新升级

非卖品，仅供非商业用途或交流学习使用
严禁网络传播本PDF，违者责任自负！
版权所有，严禁传播，违者自负法律责任！

POSTS & TELECOM PRESS

非卖品，仅供非商业用途，版权所有

作者简介

刘明洋，国内知名实战派 iOS 专家，拥有雄厚的 iOS 开发实力，是国内较早从事 iOS 开发人士之一，项目实践经验丰富。著有《Swift 语言实战精讲》一书。计算机专业毕业后一直从事软件开发和管理工作。曾开发过武汉轻轨 1 号线 LED 乘客引导系统、大型广场显示屏 PLC 监控系统。为企业定制了上百款 App，有几十款作品在 App Store 上架。具有多年 Web、软件、iOS 开发经验。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

开课吧 kaikeba.com 慧科研究院 Huikedu Research 高校邦 gaoxiaobang.com

中国教育创新校企联盟专家委员会主任 陈滢

中国计算机学会教育工作委员会委员 管刚

慧科集团技术副总裁 李嘉

联合推荐

微信小程序实战入门（第2版）

刘明洋 著 汪鸿俊 任小蕾 审

人民邮电出版社

北京

非卖品，仅供非商业用途或交流学习使用

版权相关注意事项：

1、书籍版权归著者和出版社所有

2、本PDF来自于各个广泛的信息平台，经过整理而成

3、本PDF仅限用于非商业用途或者个人交流研究学习使用
4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负
5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
7、请于下载PDF后24小时内研究使用并删掉本PDF

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

图书在版编目（CIP）数据

微信小程序实战入门 / 刘明洋著. -- 2版. -- 北京：人民邮电出版社，2018.11
ISBN 978-7-115-49328-6

Ⅰ. ①微… Ⅱ. ①刘… Ⅲ. ①移动终端－应用程序－程序设计 Ⅳ. ①TN929.53

中国版本图书馆CIP数据核字(2018)第208803号

内 容 提 要

本书不但讲解了开发工具的使用、框架的使用和微信 API 接口等相关知识点，更是结合 CSS 知识、HTML 知识带领大家去解决实际开发中的一些问题。第七章使用不同应用场景的小程序，通过完整案例来讲解小程序的实战开发。使读者不但能了解语法、函数方法、模块、事件交互、组件的应用与开发、API 等知识点，更能真正解决实际开发中遇到的一些问题，能够让读者快速掌握小程序项目的开发，并通过一本书玩转小程序开发。

本书详细介绍微信小程序开发涉及的内容和关键技术，帮助开发者快速掌握小程序开发，主要包括界面、网络、本地数据及缓存、设备硬件、微信开发接口、媒体、后端开发与设计。本书适用于希望从事软件开发行业的学生、iOS 与 Android 开发人员以及想要快速进行实际小程序项目开发的读者。

◆ 著　　刘明洋
审　　汪鸿俊　任小蕾

责任编辑　周　璇
责任印制　彭志环
◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
固安县铭成印刷有限公司印刷
◆ 开本：800×1000　1/16
印张：31.25　　2018 年 11 月第 2 版
字数：684 千字　　2018 年 11 月河北第 1 次印刷

定价：89.00 元

读者服务热线：(010)81055339　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

推荐语

微信小程序带来了基于社交平台的移动应用新范式，激发了许多应用创新，引起了广大开发者的极大关注。本书概念介绍深入浅出，通俗易懂，案例丰富且剖析彻底，火候拿捏得当，足见作者在程序设计方面理论和实践深度结合的功底。我相信这本书一定会成为读者的一本关于微信小程序的大百科。

——陈滢
中国教育创新校企联盟专家委员会主任
中国电子学会物联网专家委员会委员
中国计算机学会（CCF）高级会员
CCF YOCSEF（中国计算机学会青年计算机科技论坛）委员
教育部“互联网+大学生创新创业大赛”专家与国赛评委
IBM 中国研究院前院长
广东省云计算协会特聘专家
海南省信息化建设评审专家
慧科集团创始合伙人、慧科集团首席战略官

2017 年 1 月，新年伊始，一个新鲜事物——微信小程序万众瞩目。在微信生态下，触手可及的微信小程序引起广泛关注，用户不用再担心安装太多眼花缭乱的应用而占用手机更多内存的问题，因为小程序具有随时可用，但又无须安装、卸载的特点。

这本书虽然不是小程序相关的第一本书，但一定是内容全面、理论结合实践并且通俗易懂的一本书。

——管刚
中国计算机学会（CCF）高级会员

CCF 教育工作委员会委员
CCF 青年计算机科技论坛学术委员，并多次被评为 CCF 杰出演讲者
慧科集团合伙人、集团高级副总裁

微信在今天的热度大家有目共睹，已成为互联网营销模式一条必经之路，当然无数成功的营销结果证明这是一条正确的路。而小程序作为信息传播和搜索的又一个有效平台，必将

非卖品，仅供非商业用途或交流学习使用
版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

互联网营销模式带入一个新的高度——谨将刘明洋的这本书推荐给大家，让我们一起走进小程序的时代。

——李嘉
曾经效力于 Sybase、Sun Microsystem、SAS 等国际知名软件公司
慧科集团技术副总裁

非卖品，仅供非商业用途或交流学习使用

版权相关注意事项：
1、书籍版权归著者和出版社所有
2、本PDF来自于各个广泛的信息平台，经过整理而成
3、本PDF仅限用于非商业用途或者个人交流研究学习使用
4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负
5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
7、请于下载PDF后24小时内研究使用并删掉本PDF

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

前言

2017 年 1 月 9 日凌晨，“小程序”终于揭开神秘面纱，正式上线。小程序是微信继订阅号、服务号、企业号之后推出的一种新的并行体系，不需要下载、安装即可使用的应用。它实现了应用“触手可及”的梦想，用户扫一扫或者搜一下即可打开应用，也体现了“用完即走”的理念，随时可用，但又无须安装、卸载。微信小程序可以为开发者提供基于微信的表单、导

航、地图、媒体和位置等开发组件，让他们在微信的网页里构建一个 HTML5 应用。同时微信还开放了登录和微信支付等接口，让这个“小程序”可以和用户的微信账号打通。微信将“小程序”定义为“一种新的应用形态”。

将微信小程序开发与原生 App（应用）开发相对比，超过 95%的原生应用功能如果使用微信小程序开发都会变得更容易、更快捷。除了开发效率更高，它还具有更好的兼容性，无论是 iOS 还是 Android 又或者是 WP，都能很好地工作，真正做到一次开发随处可用。微信小程序能获得更多的系统权限。首先是数据缓存能力，这可以让用户在打开一个小程序的时候将程序的主要框架缓存到微信上，下一次就可以快速打开。

微信小程序具备更好的可维护性。传统的移动应用开发，如需新增功能，开发者只能重新编译应用程序并上传新版本，用户需更新后才能使用新的功能，若用户不更新或更新不及时，则容易造成多版本系统并行运行的问题，给开发、运维带来较多麻烦。微信小程序具有更好的传播性、更低的获取用户成本。传统的 App 从十几兆到百兆大小不等且会占用手机内存，用户从下载到使用再到卸载都需要时间投入，这些都给 App 的推广以及用户的使用带来了不便。使用微信小程序，在任何安装了微信的手机中都可以运行。直接搜索或扫描二维码可以让你快速打开小程序，用完即关。在 Android 手机中，它可以和 Android 本地应用一样，在手机桌面创建应用程序图标。除了具备普通移动应用的功能外，它还能利用微信向用户推送消息，利用微信完成支付，在某种程度上顺应了方便、快捷的互联网传播形式，随着用户需求的发展，微信还将开放更多的功能提供给开发者。

小程序开发人员的需求量会在短时间内激增。尽早加入小程序开发者的行列，在供小于求的时间点入行，可以增加更多的个人价值，成为一个热门抢手的程序员。“工欲善其事，必先利其器”，如果你想开发小程序，必须先学会一套微信特制的“开发语言”。虽然官方有开发文档介绍，但是对于很多普通人、新手来说，在学习的过程中常常会遇到各种各样的问题，还需要具备 HTML、JavaScript、CSS 等基本的 Web 前端能力。为了更快地上手这门开发语言，需要学习一些 HTML、JavaScript、CSS 相关教程。而本书结合多个项目实例详细介绍了 CSS 样式的使用以及小程序的完整开发过程，可谓一书在手，开发不愁！

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

本书特点

- 容易上手：本书通过案例精细讲解小程序语言的实战技巧，使读者容易理解，并能马上学以致用。对于每一部分具体内容，本书都精心设计了相应的示例程序，一方面可以帮助读者加深理解，另一方面也可以逐步培养读者的程序设计能力。
- 内容全面：本书详细讲解了开发工具的使用、基本组件、API 的使用。
- 技术实用：本书通过多个案例详细讲解了小程序的开发过程和代码实现，让读者从 0 到 1 开发属于自己的小应用。

本书内容

- 第一章介绍了如何注册开发者、开发环境的安装和使用，带领大家创建第一个项目。

● 第二章详细介绍了小程序的应用场景、全局配置、架构以及小程序的调试、上传和发布等。

● 第三章完整介绍了小程序的框架组件，包括视图容器、基础内容、表单组件、操作反馈、导航、媒体组件、地图、画布、客服会话等。

● 第四章介绍了所有 API 的使用，主要包括网络、媒体、文件、数据缓存、位置、设备、界面交互、绘图、拓展接口等。

● 第五章介绍了开放 API 的使用，例如：登录、签名加密、授权、用户信息、微信支付、模板消息、客服消息、分享、二维码、收货地址、卡券、设置等。

● 第六章介绍了数据统计分析、小程序版本更新，以及多线程的使用。

● 第七章通过几个实际项目案例来讲解小程序的开发过程和代码实现。带领大家从 0 到 1 实现自己的小程序。主要包括仿新闻小应用、书架功能、录音功能、二维码生成器、图片滤镜、仿电影小应用。

读者对象

本书既可以作为相关院校微信小程序开发教学的专业教材，又可以作为想进入或者已进入微信小程序开发行列的人员使用的参考书。具体目标读者定位为：

（1）希望从事软件开发行业的学生；

（2）想开发小程序的 iOS 与 Android 开发人员；

（3）从事 ASP.NET、JSP、PHP 等工作的程序员；

（4）想要快速进行实际小程序项目开发的读者；

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

（5）相关培训机构的老师和学员；

（6）程序测试及维护人员；

（7）编程爱好者；

（8）参加实习的初级程序员；

（9）大中专院校的老师和学生；

（10）初中级程序开发人员。

本书的内容能够帮助读者提升实际的工作能力。

致谢

非常感谢人民邮电出版社的编辑对本书的帮助和支持。

非常感谢中国教育创新校企联盟专家委员会主任陈滢博士，慧科集团合伙人、集团高级副总裁管刚，曾经效力于 Sybase、Sun Microsystem、SAS 等国际知名软件公司，目前在慧科集团任集团技术副总裁的李嘉对本书的大力支持。

感谢开课吧合伙人、无限互联联合创始人、国内培训界著名导师——汪鸿俊老师。

另外也非常感谢以下家人、朋友对本书的大力帮助和支持，他们分别是：江本涛、蒋仪菊、刘全利、徐次田、徐侠田、任小蕾、赵藤安、徐海田、付光琪、蒋峰。

联系方式

希望本书能够帮助大家去创建属于自己的小程序，由于时间仓促，本书难免存有疏漏，敬请读者指正，欢迎对我们的工作提出建议，谢谢。

请通过下面的方式联系我们：

微信小程序爱好者 QQ 群：248728021

加入微信小程序爱好者 QQ 群，一起探讨、一起探究、一起成为小程序的专业开发者。

代码获取方式

请联系 QQ 群 248728021，获取最新源代码。

——刘明洋

非卖品，仅供非商业用途或交流学习使用

版权相关注意事项：

1、书籍版权归著者和出版社所有

2、本PDF来自于各个广泛的信息平台，经过整理而成

3、本PDF仅限用于非商业用途或者个人交流研究学习使用

4、本PDF获得者不得在互联网上以任何目的进行传播，违规者造成的法律责任和后果，违规者自负

5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF

6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅

7、请于下载PDF后24小时内研究使用并删掉本PDF

目 录

第一章 创建项目 …… 1

1.1 注册小程序开发者 …… 1
1.2 安装环境 …… 7
1.3 创建小程序 …… 7
1.4 开发工具介绍 …… 14
1.4.1 概览 …… 14
1.4.2 编辑器 …… 16
1.4.3 调试器 …… 22
1.4.4 项目详情 …… 28
1.4.5 运行预览 …… 29

第二章 小程序详细介绍 …… 31

2.1 小程序、原生 App、WebApp 的区别 …… 31
2.2 应用场景 …… 34
2.3 全局配置（app.json）和页面配置（*.json） …… 35
2.3.1 全局配置 app.json 详解 …… 35
2.3.2 页面配置（*.json） …… 40
2.4 小程序架构 …… 40
2.4.1 框架介绍 …… 40
2.4.2 逻辑层 …… 42
2.4.3 视图层（WXML 和 WXSS 介绍） …… 48
2.4.4 数据层 …… 49
2.5 视图层 WXML 介绍 …… 58
2.6 视图层 WXSS 介绍 …… 71
2.7 WXML 与 HTML 的区别 …… 73
2.8 小程序调试、上传、发布 …… 74
2.8.1 事前准备：HTTPS …… 74
2.8.2 预览及调试 …… 75
2.8.3 发布 …… 77

第三章 框架组件 …… 81

3.1 框架组件介绍 …… 81
3.2 视图容器 …… 84
3.2.1 view …… 84
3.2.2 scroll-view …… 103
3.2.3 swiper 与 swiper-item …… 108
3.2.4 movable-area 与 movable-view …… 114
3.2.5 cover-view 与 cover-image …… 115
3.3 基础内容 …… 117
3.3.1 icon …… 117
3.3.2 text …… 120
3.3.3 rich-text …… 123
3.3.4 progress …… 125
3.4 表单组件 …… 126
3.4.1 button …… 126
3.4.2 checkbox 与 checkbox-group …… 129
3.4.3 form …… 131

3.4.4 input …… 133
3.4.5 label …… 140
3.4.6 picker …… 143
3.4.7 picker-view …… 152
3.4.8 radio …… 156
3.4.9 slider …… 158
3.4.10 switch …… 160
3.4.11 textarea …… 163
3.5 操作反馈 …… 166
3.5.1 action-sheet …… 166
3.5.2 modal …… 169
3.5.3 toast …… 171
3.5.4 loading …… 173
3.6 导航（navigator） …… 174
3.7 媒体组件 …… 178
3.7.1 audio …… 178
3.7.2 image …… 181
3.7.3 video …… 185
3.7.4 camera …… 190
3.8 地图（map） …… 191
3.9 画布（canvas） …… 197
3.10 客服会话按钮（contact-button） … 199
3.11 开放数据（open-data） …… 199

第四章 API …… 201

4.1 网络 …… 201
4.1.1 wx.request（OBJECT）发起请求 …… 202
4.1.2 上传、下载 …… 205
4.1.3 Websocket …… 209
4.2 媒体 …… 214
4.2.1 图片 …… 214
4.2.2 录音 …… 219
4.2.3 语音播放控制 …… 220
4.2.4 音乐播放控制 …… 221
4.2.5 视频和视频组件控制 …… 229
4.3 文件 …… 233
4.4 数据缓存 …… 238
4.5 位置 …… 243
4.5.1 获取位置 …… 243
4.5.2 查看位置 …… 245
4.5.3 地图组件控制 …… 246
4.6 设备 …… 248
4.6.1 系统信息 …… 248
4.6.2 网络状态 …… 252
4.6.3 重力加速度 …… 252
4.6.4 罗盘 …… 254
4.6.5 拨打电话 …… 256
4.6.6 扫码 …… 256
4.6.7 剪贴板 …… 257
4.6.8 蓝牙 …… 258
4.6.9 iBeacon …… 273
4.6.10 屏幕亮度 …… 276
4.6.11 用户截屏事件 …… 278
4.6.12 振动 …… 278
4.6.13 手机联系人 …… 279
4.6.14 NFC …… 281
4.6.15 Wi-Fi …… 284
4.7 界面交互 …… 289
4.7.1 交互反馈 …… 289
4.7.2 导航和 tabBar 的使用介绍 …… 292
4.7.3 设置 tabBar …… 298
4.7.4 设置置顶信息 …… 301
4.7.5 动画 …… 302
4.7.6 下拉刷新 …… 306
4.7.7 位置 …… 307
4.8 绘图 …… 310
4.8.1 坐标系介绍

（coordinates）……311
4.8.2 绘图主接口……312
4.8.3 填充颜色、线条、阴影……315
4.8.4 渐变……316
4.8.5 线条样式……318
4.8.6 矩形……323
4.8.7 路径……325
4.8.8 变形……333
4.8.9 文字（设置字号/绘制文本）……334
4.8.10 图片（drawImage）……337
4.8.11 全局画笔透明度（setGlobalAlpha）……337
4.8.12 其他……338
4.9 拓展接口……343
4.10 调试接口……343
4.11 WXML 节点信息……344
4.12 WXML 节点布局相交状态……351
4.13 第三方平台……353

第五章 开放 API……355

5.1 登录……355
5.2 签名加密……360
5.3 授权……362
5.4 用户信息……363
5.5 微信支付……365
5.6 模板消息……367
5.6.1 使用说明……367
5.6.2 接口说明……368
5.7 客服消息……371
5.7.1 接收消息和事件……371
5.7.2 发送客服消息……374
5.7.3 临时素材接口……376
5.7.4 接入指引……377
5.7.5 转发消息……379
5.7.6 客服输入状态……380
5.8 转发（分享）……381
5.9 二维码……386
5.10 收货地址……387
5.11 卡券……388
5.12 设置……390
5.13 微信运动……392
5.14 打开小程序……393
5.15 打开 App……394
5.16 获取发票抬头……396
5.17 生物认证……397
5.18 附近……398
5.19 插件管理……403

第六章 数据、更新、多线程……408

6.1 数据……408
6.1.1 概况趋势……408
6.1.2 访问趋势……409
6.1.3 访问分布……412
6.1.4 访问留存……417
6.1.5 访问页面……420
6.1.6 用户画像……423
6.1.7 自定义分析……425
6.2 更新……426
6.3 多线程……428

第七章 项目实战……431

7.1 仿新闻小应用……431
7.1.1 通过 tabBar 实现页面之间的切换……432
7.1.2 顶部滑动菜单的实现……433
7.1.3 新闻列表的实现……436
7.1.4 首页完整代码……437

7.1.5 用户中心界面实现 … 444
7.1.6 用户中心界面完整代码 … 446
7.2 书架功能 … 450
7.2.1 精彩推荐模块实现 … 451
7.2.2 热门书籍模块实现 … 451
7.2.3 精品书籍模块实现 … 452
7.3 录音功能 … 458
7.4 二维码生成器 … 464
7.5 图片滤镜 … 466
7.5.1 模糊、怀旧、复古、美白功能的实现 … 466
7.5.2 饱和度、亮度、对比度功能的实现 … 469
7.5.3 动态滤镜的实现 … 470
7.6 仿电影小应用 … 471
7.6.1 电影列表页面的实现 … 472
7.6.2 电影详情页面的实现 … 478
7.6.3 搜索页面的实现 … 481
7.7 总结 … 488

第一章

创建项目

学习微信小程序开发，首先要从了解开发环境开始，本章将主要从注册小程序开发者开始讲述如何安装环境，以及带领大家创建小程序，并详细介绍开发工具的使用方法。本书中出现的二维码仅为演示使用，故特意做模糊处理，仅供参考。

1.1 注册小程序开发者

首先要注册微信公众平台开发者，打开微信公众平台官网首页，单击右上角的“立即注册”按钮，如图 1-1 所示。

微信公众平台账号类型目前分为 4 类（订阅号、服务号、小程序、企业微信）。选择“小程序”，如图 1-2 所示，单击“账号类型区别”可查看不同类型账号的区别和优势。

图 1-1 微信公众平台官网

图 1-2 注册账号类型

选择小程序之后，填写邮箱和密码，这里邮箱需要填写未注册，未绑定过开放平台、订阅号、服务号、企业微信的邮箱，如图 1-3 所示。

填写信息提交之后，系统会发送激活邮件到邮箱进行确认，如图 1-4 所示。

登录邮箱，查收激活邮件，单击激活链接，如图 1-5 所示。

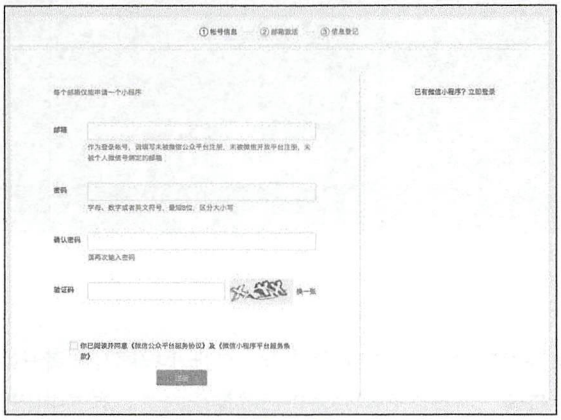


图 1-3　账号信息

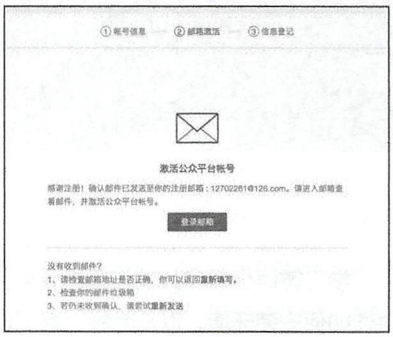


图 1-4　邮箱激活

单击激活链接后，继续下一步的注册流程，如图 1-6 所示。请选择主体类型，完善主体信息和管理员信息。确认微信公众账号主体类型属于政府、媒体、企业、其他组织、个人，并按照对应的类别进行信息登记。

图 1-5　激活邮件内容

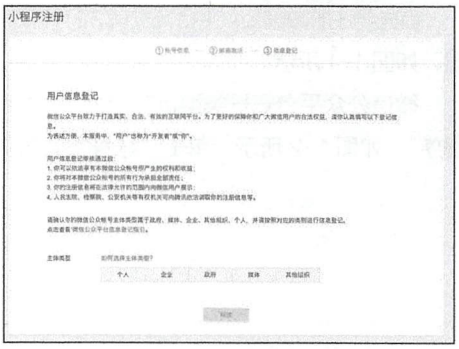


图 1-6　信息登记

主体类型说明如表 1-1 所示。

表 1-1　　主体类型说明

账号主体	范围
个人	18 岁以上有国内身份信息的微信实名用户
企业	企业、分支机构、企业相关品牌
企业（个体工商户）	个体工商户
政府	国内各级、各类政府机构、事业单位、具有行政职能的社会组织等。目前主要覆盖公安机构、党团机构、司法机构、交通机构、旅游机构、工商税务机构、市政机构等
媒体	报纸、杂志、电视、电台、通讯社、其他
其他组织	不属于政府、媒体、企业或个人的类型

个人类型账号包括：由自然人注册和运营的公众账号。个人类型账号暂不支持微信认证、微信支付及高级接口功能。

为了验证你的身份，请用绑定了管理员本人银行卡的微信扫描二维码。本验证方式不扣除任何费用。

注册后，扫码的微信号将成为该账号的管理员微信号。若微信没有绑定银行卡，请先绑定。个人主体登记信息如图 1-7 所示。

企业类型账号可选择两种主体验证方式，如图 1-8 所示。方式一：需要用该对公账户向腾讯公司进行打款来验证主体身份。打款信息在提交主体信息后最后一步可以看到。方式二：微信注册并认证，无须小额打款验证，需支付 300 元人民币审核费用。提交认证后会在 1~5 个工作日完成审核。在认证完成前小程序部分功能暂不支持。

图 1-7 个人类型账户

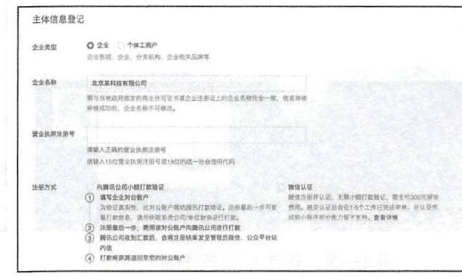


图 1-8 企业类型账户

政府、媒体、其他组织类型账号如图 1-9、图 1-10、图 1-11 所示，必须通过微信认证验证主体身份。认证通过前，小程序部分功能暂无法使用。

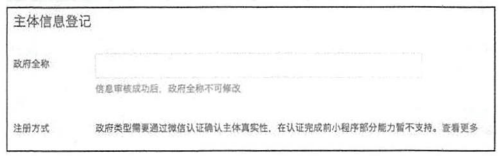


图 1-9 政府类型账户

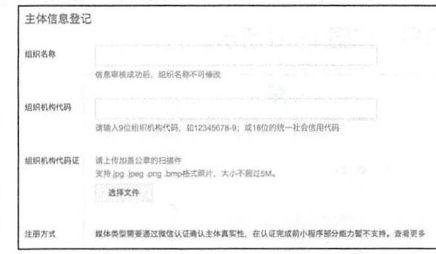


图 1-10 媒体类型账户

填写管理员信息，如图 1-12 所示。确认主体信息不可变更，单击确认完成注册流程。

完成注册后，登录小程序管理平台，在微信公众平台官网首页登录，如图 1-13 所示。需要注意，这里登录之后需要管理员使用微信客户端扫描确认才可以登录成功。

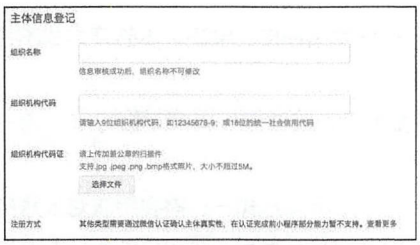


图 1-11　其他组织类型账户

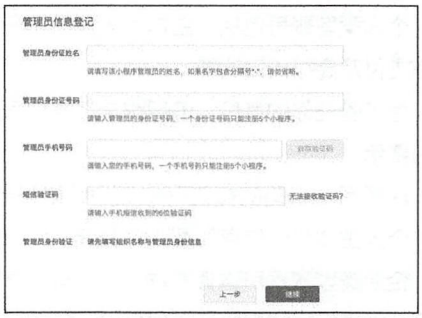


图 1-12　管理员信息登记

在小程序平台首页，左侧列表最底部，单击“设置”，找到“微信认证”后面的详情，如图 1-14 所示。

图 1-13　登录小程序平台

图 1-14　微信认证入口

选择微信认证验证主体身份的用户，完成注册流程后需要尽快进行微信认证，认证完成之前部分功能暂不可使用。选择对公打款的用户，可以根据页面提示，向指定的收款账号汇入指定金额，如图 1-15 所示。需要在 10 天内完成汇款，否则注册将失败。

微信认证之后，用户可以通过“首页”小程序发布流程下面的“小程序信息”来完善小程序信息。如图 1-16 所示。

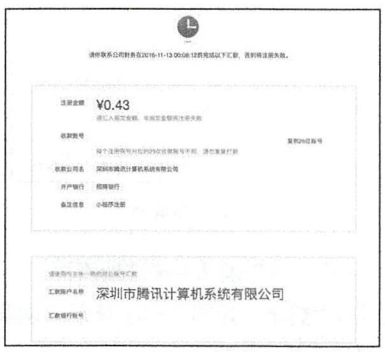


图 1-15　对公打款

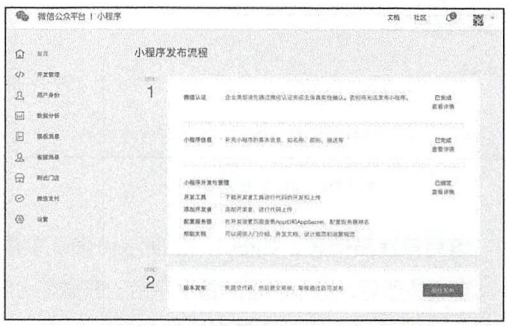


图 1-16　补充小程序信息位置示意图

用户可以补充小程序名称、图标、描述等，并选择服务范围，如果选择了特殊行业的服务类目，需要提供资质材料提交审核，如图 1-17 所示。

单击左侧菜单栏里面的“用户身份”，再单击开发者，如图 1-18 所示，可以绑定开发者。已认证的小程序可以最多绑定 20 个开发者，未认证的小程序可以最多绑定 10 个开发者。

图 1-17 补充小程序信息

图 1-18 开发者列表

单击“绑定”，可以绑定新的开发者，这里需要管理员通过微信客户端扫码确认才可以绑定开发者，如图 1-19 所示。输入微信号，即可绑定开发者，如图 1-20 所示。

图 1-19 管理员扫码

图 1-20 绑定开发者

单击左侧菜单栏里面的“设置”，再单击“开发者设置”，就可以获取 AppID，如图 1-21 所示。没有 AppID 也可以进行小程序的开发练习，只是部分功能受限制，而且不能上传发布。并且只有管理员和绑定的开发者才有权限在手机上体验该 AppID 的小程序。

个人没有注册微信小程序账号的，可以通过已有的订阅号（或新注册订阅号）来开发小程序。使用订阅号登录微信公众平台，在左侧菜单栏中选择“开发”栏目下面的“基本设置”，单击“成为开发者”，如图 1-22 所示。

图 1-21　AppID 信息

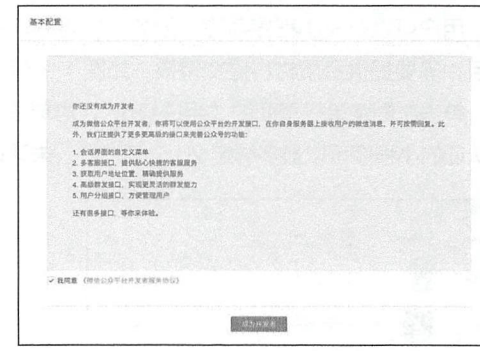


图 1-22　开通开发者

然后在左侧菜单栏中选择“开发”栏目下面的“开发者工具”，找到“web 开发者工具”单击“进入”，如图 1-23 所示。单击“绑定开发者微信号”，如图 1-24 所示，用户可以通过微信号、qq 号、手机号来搜索绑定的开发者微信号，这里需要注意的是所绑定的用户必须关注该订阅号，否则不能绑定成开发者。如图 1-25 所示。

图 1-23　开发者工具

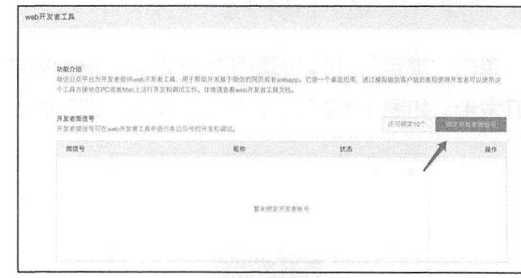


图 1-24　开发者管理

图 1-25　绑定开发者微信号

单击“邀请绑定”之后，开发者手机的微信会收到一条消息，需要接受邀请，才会成为真正的开发者。

1.2 安装环境

截止到 2018 年 8 月 3 日，微信团队提供的开发者工具最新版本是 2018.08.03 (1.02.1808030)，有Windows 64、Windows 32、Mac 三种版本。下载地址是：微信公众平台—小程序开发—工具，用户可以根据自己需求下载想要的版本。

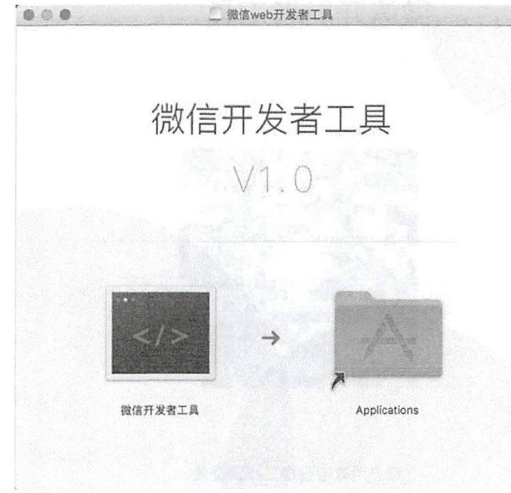


图 1-26 微信开发工具安装界面

这里我们以 Mac 版本安装包为例，介绍一下安装过程。

单击“Mac”，浏览器会自动下载安装软件，等下载完成之后，双击“wechat_devtools_1.02.1808030.dmg”安装包，将出现安装界面，如图 1-26 所示。

在 Mac 平台下安装软件相对比较简单，可以直接把安装文件手动拖到“Applications”文件夹内，复制完成就实现了安装。在 Windows 平台可以按照提示一步步进行安装。

1.3 创建小程序

安装完毕之后，在应用程序里面找到微信开发者工具，双击运行。首次打开微信开发者工具，可以看到两种开发模式的选择。

1. 小程序调试。选择小程序调试，将进入小程序本地项目管理页，可以新建、删除本地的项目，也可以选择本地已存在的项目。

2. 公众号网页调试。选择公众号网页调试，将直接进入公众号网页项目调试界面，在地址栏输入 URL，即可调试该网页的微信授权以及微信 JS-SDK 功能。

单击程序项目，这时需要管理员或者绑定的开发者使用手机微信扫码登录，如图 1-27 所示。

扫码登录之后，可以看到如图 1-28 所示的界面。

单击“小程序项目”，进入项目列表界面，如图 1-29 所示。在这里可以创建新的项目，也可以打开历史创建的项目。

单击“管理项目”可以删除历史项目记录，但不会删除硬盘上的项目文件。单击“+”后选择本地空文件夹作为项目目录，输入小程序的 AppID（没有 AppID 的可以注册或选择体验小程序、小游戏学习小程序开发），输入项目名称，如图 1-30 所示。注意：“项目目录”可以

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

选择本地已有的小程序项目，单击“确定”即可完成微信小程序项目的导入。

图 1-27 微信开发工具登录界面

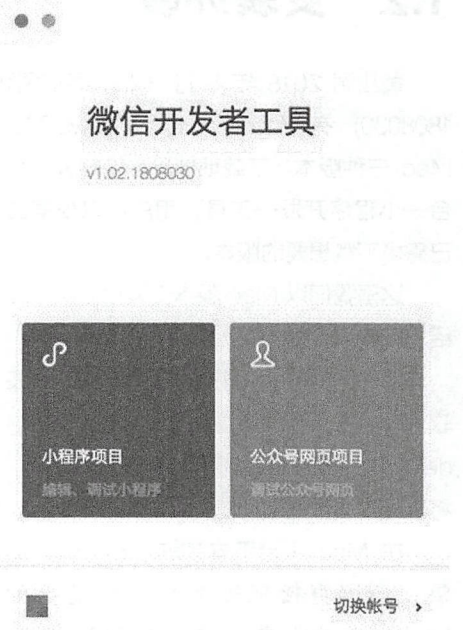


图 1-28 微信开发工具首页

图 1-29 小程序项目列表

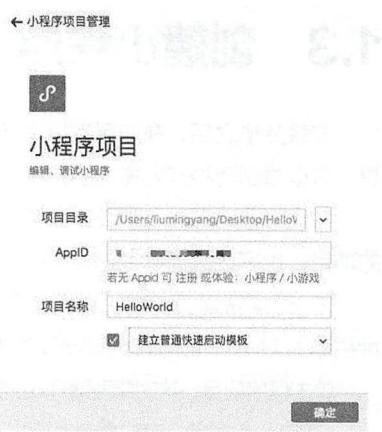


图 1-30 添加项目

勾选“建立普通快速启动模板”，创建的项目带有 pages、utils 文件夹和 app.js、app.json、app.wxss 和 project.config.json 文件，如图 1-31 所示。

第一个微信小程序项目“HelloWorld”就创建成功了。此时可以看到完整的开发者工具界面，如图 1-32 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

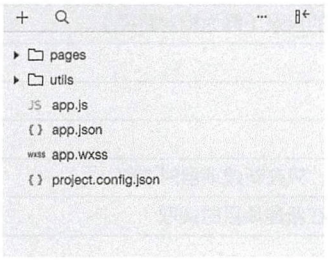


图 1-31 勾选“建立普通快速启动模板”的项目结构

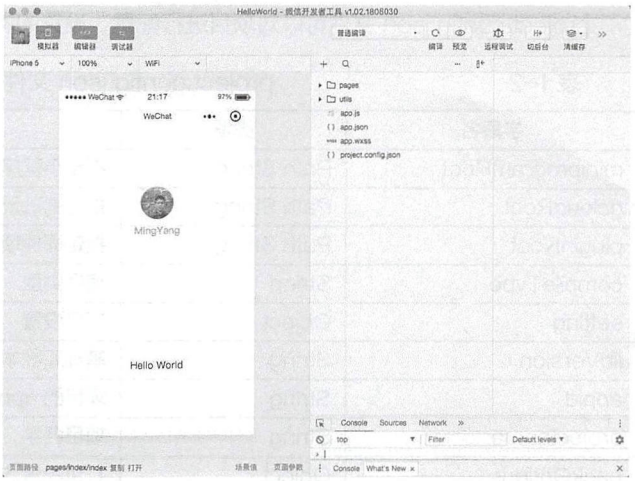


图 1-32 开发者工具界面

勾选“建立插件快速启动模板”可以创建小程序插件项目。开发者可以通过该功能，强化自身小程序能力；小程序服务提供商也可以用它，为开发者、用户提供强大的小程序功能支持，进一步拓展小程序能力。

小程序开发者无须重新注册账号，即可直接在小程序管理后台开通插件功能，完成基本信息填写后完成开通。开通入口：小程序管理后台-小程序插件。

插件是独立于小程序之外的，但是 AppID 是公用的，所以不要使用原有的小程序项目进行插件开发。在创建项目页面，选择一个空文件夹作为项目路径，否则不会显示“建立插件快速启动模板”选项。

插件项目的结构包括 miniprogram、plugin、project.config.json，如图 1-33 所示。

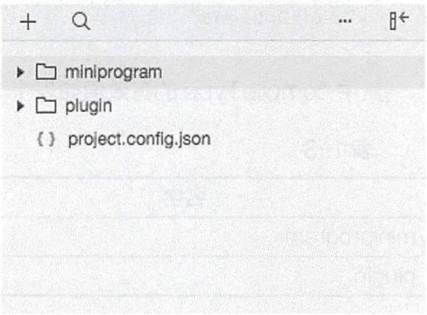


图 1-33 插件项目结构

结构说明：

1. miniprogram 文件夹是一个普通小程序项目，用来编写小程序插件使用的 Demo，上传插件代码时这个 Demo 会一起上传，并作为小程序插件发布的审核依据。

2. plugin 文件就是小程序插件项目，用来编写小程序插件的代码。

3. project.config.json 需要关注 compileType 字段，当 compileType == 'plugin' 时才能正常地使用插件项目。在项目开发期间，可以手动修改 project.config.json 文件的 compileType 字段来切换项目的编译类型。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

project.config.json 文件可以对项目进行配置，设置的属性如表 1-2 所示。

表 1-2 project.config.json 文件使用的属性

字段名	类型	说明
miniprogramRoot	Path String	指定小程序源码的目录（需为相对路径）
qcloudRoot	Path String	指定腾讯云项目的目录（需为相对路径）
pluginRoot	Path String	指定插件项目的目录（需为相对路径）
compileType	String	编译类型
setting	Object	项目设置
libVersion	String	基础库版本
appid	String	项目的 appid，只在新建项目时读取
projectname	String	项目名字，只在新建项目时读取
packOptions	Object	打包配置选项
scripts	Object	自定义预处理

项目配置示例：

```
{
  "miniprogramRoot": "./src",
  "qcloudRoot": "./svr",
  "setting": {
    "postcss": true,
    "es6": true,
    "minified": true,
    "urlCheck": false
  },
  "packOptions": {}
}
```

其中 compileType 的有效值如表 1-3 所示。

表 1-3 compileType 的有效值

名字	说明
miniprogram	当前为普通小程序项目
plugin	当前为小程序插件项目

setting 中可以指定以下设置，如表 1-4 所示。

表 1-4 setting 中可以指定的设置

字段名	类型	说明
es6	Boolean	是否启用 es5 转 es6
postcss	Boolean	上传代码时样式是否自动补全
minified	Boolean	上传代码时是否自动压缩
urlCheck	Boolean	是否检查安全域名和 TLS 版本

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

scripts 中指定自定义预处理的命令，如表 1-5 所示。

表 1-5 scripts 中指定的命令

名字	说明
beforeCompile	编译前预处理命令
beforePreview	预览前预处理命令
beforeUpload	上传前预处理命令

packOptions 用以配置项目在打包过程中的选项。打包是预览、上传时对项目进行的必须步骤。

目前可以指定 packOptions.ignore 对象，用以配置打包时对符合指定规则的文件或文件夹进行忽略，以跳过打包的过程，这些文件或文件夹将不会出现在预览或上传的结果内。

packOptions.ignore 为一个对象数组，对象元素类型如表 1-6 所示。

表 1-6 packOptions.ignore 数组元素

字段名	类型	说明
value	String	路径或取值
type	String	类型

其中，type 可以取的值为 folder、file、suffix、prefix，分别对应文件夹、文件、后缀、前缀。

value 字段的值不支持通配符、正则表达式。若表示文件或文件夹路径，以小程序目录（miniprogramRoot）为根目录。示例配置如下：

```
{
  "packOptions": {
    "ignore": [{
      "type": "file",
      "value": "test/test.js"
    }, {
      "type": "folder",
      "value": "test"
    }, {
      "type": "suffix",
      "value": ".webp"
    }, {
      "type": "prefix",
      "value": "test-"
    }]
  }
}
```

插件 API 开发示例讲解，以开发一个 API 接口，返回“Hello weixin”为例。

第 1 步：找到 plugin/api/data.js 增加 sayHello 方法，module.exports 中暴露 sayHello，具体代码如下所示：

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
module.exports = {
  getData: getData,
  setData: setData,
  sayHello
}

function sayHello() {
  return 'Hello weixin'
}
```

第 2 步：在 plugin/index.js 中暴露出需要给插件使用者使用的接口，代码如下所示：

```
var data = require('./api/data.js')

module.exports = {
  getData: data.getData,
  setData: data.setData,
  sayHello: data.sayHello
}
```

第 3 步：在 plugin/plugin.json 的配置文件中，配置插件的入口，默认代码如下所示：

```
{
  "publicComponents": {
    "list": "components/list/list"
  },
  "main": "index.js"
}
```

第 4 步：在 miniprogram 中使用该接口。打开 miniprogram/pages/index/index.js 使用该接口，代码如下所示：

```
var plugin = requirePlugin("myPlugin")
Page({
  onLoad: function() {
    plugin.getData()
    console.log(plugin.sayHello())
  }
})
```

其中 myPlugin 为我们的插件名，miniprogram 项目默认配置可以在 miniprogram/app.json 中修改，代码如下所示：

```
{
  "pages": [
    "pages/index/index"
  ],
  "plugins": {
    "myPlugin": {
      "version": "dev",
      "provider": "wx3602ec25ae74cfea"
    }
  }
}
```

插件组件开发示例讲解，以开发一个组件，组件显示“Hello weixin”为例。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

第 1 步：在 plugin/components 下新建一个目录 HelloWX，单击该文件夹，右键单击并选择“新建 Components”，输入 HelloWX，与普通组件一样，将生成以下 4 个文件，如图 1-34 所示。

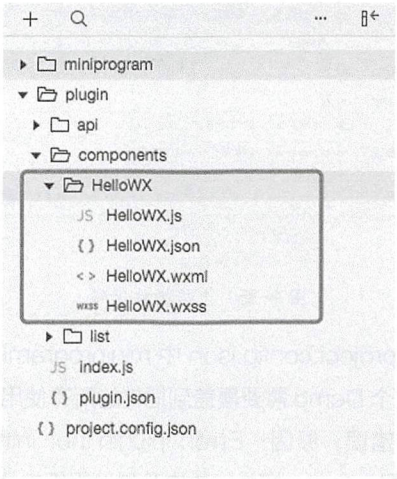


图 1-34 HelloWX 组件

第 2 步：在 helloWorld.wxml 中编写视图代码，代码如下所示：

```
<!--plugin/components/hello/hello.wxml-->
<text>Hello weixin </text>
```

第 3 步：在 plugin/plugin.json 中配置需要暴露给插件使用者使用的组件：

```
{
  "publicComponents": {
    "list": "components/list/list",
    "HelloWX": "components/HelloWX/HelloWX"
  },
  "main": "index.js"
}
```

第 4 步：在需要引用该组件的页面的配置文件中，做好配置即可，例如在 index 页面使用，则需要在 miniprogram/pages/index/index.json 中进行如下配置：

```
{
  "usingComponents": {
    "list": "plugin://myPlugin/list",
    "HelloWX": "plugin://myPlugin/HelloWX"
  }
}
```

其中 HelloWX 为我们的组件，miniprogram/pages/index/index.wxml 使用“<HelloWX />”即可显示自定义的组件内容。

插件开发完成后，单击开发工具的“上传”按钮，进行代码上传。上传插件代码前，需要

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

指定版本号，格式为数字.数字.数字，每个数字最大为 999。每次提交版本号需要递增，插件使用者会用到这个版本号，请谨慎填写，如图 1-35 所示。

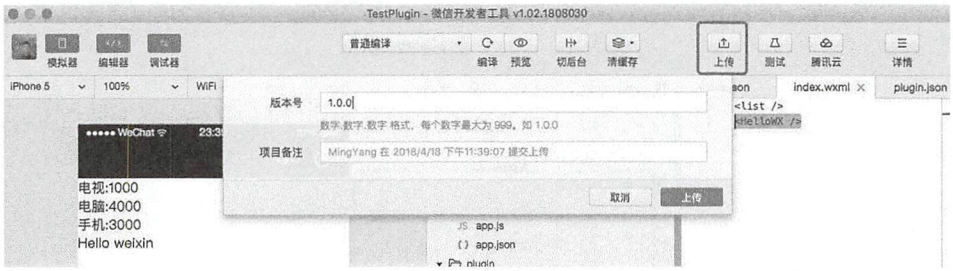


图 1-35 上传插件代码

上传插件时，同时会将 project.config.json 中 miniprogramRoot 指定的目录内容作为插件使用 Demo 一起上传，这个 Demo 需要覆盖到插件的所有使用场景，便于插件的审核。

上传时如果报错“上传错误，原因：Error: plugin not initted”，说明没有开通插件功能。目前开发插件的开放范围：企业、媒体、政府及其他组织主体的小程序。

1.4 开发工具介绍

1.4.1 概览

打开微信开发工具，我们先来熟悉一下菜单栏，主要有“项目”“文件”“编辑”“工具”“界面”和“设置”6 个功能。界面栏目主要用于设置开发工具显示的模块，主要包括：工具栏、模拟器、编辑器、目录树、调试器，如图 1-36 所示。设置栏目可以对工具的外观、快捷键、编辑、代理、通知、项目进行设置，如图 1-37 所示。

图 1-36 界面栏目

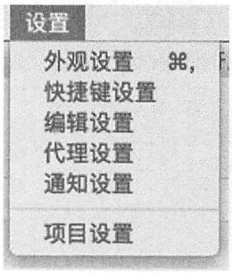


图 1-37 设置栏目

微信开发工具集成了代码编辑、编译、预览、远程调试、上传发布、测试、腾讯云等功能。主界面主要分为 5 大区域模块，如图 1-38 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

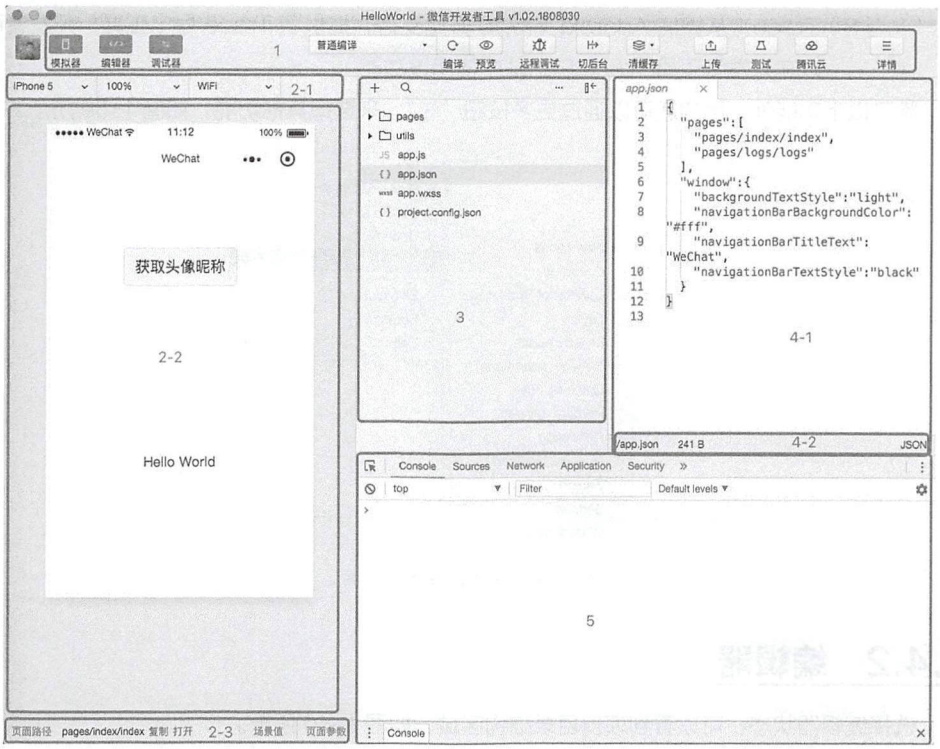


图 1-38 主界面

- 1 是导航菜单区域，用来选择使用不同的功能。其中模拟器、编辑器、调试器分别控制区域 2、3、5 的显示和隐藏。
- 2（分为 2-1、2-2、2-3）是页面实时展示区域（2-1 可以选择相应的模拟器和网络环境，2-2 为页面显示效果，2-3 为页面的相对路径）。
- 3 是项目目录结构区域。
- 4（分为 4-1、4-2）是代码编辑区域（4-1 为代码区域，4-2 为文件的相对路径）。
- 5 是调试区域，该区域主要分为：
  （1）Console（控制台，查看 log 信息）
  （2）Sources（源文件，查看项目的源代码）
  （3）Network（网络，查看网络状况）
  （4）Storage（缓存，查看本地的缓存数据。用户可以通过“导航菜单区域”的“缓存”按钮清除缓存）
  （5）AppData（App 信息）
  （6）Wxml（帮助开发者开发 Wxml 转化后的界面）
  （7）Sensor（模拟地理位置和调试重力感应）

（8）Trace（从微信 Android 6.5.10 开始支持，提供了 Trace 导出工具，开发者可以在开发者工具 Trace Panel 中使用该功能）

除了以上功能外，用户还可以通过更多按钮“⋮”，来选择其他功能，如图 1-39 所示。

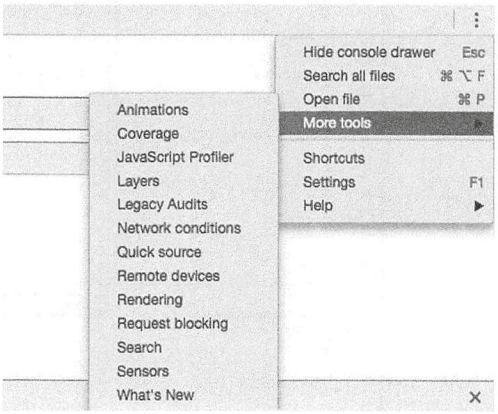


图 1-39 调试区域更多工具

1.4.2 编辑器

选择编辑器状态，可以看到项目目录结构区域，如图 1-40 所示。

小程序使用的 MINA 框架，如图 1-41 所示，目的是通过简单、高效的方式让开发者可以在微信中开发具有原生 App 体验的服务。

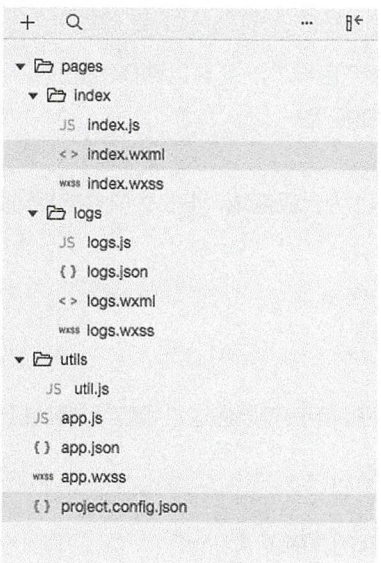


图 1-40 项目目录结构

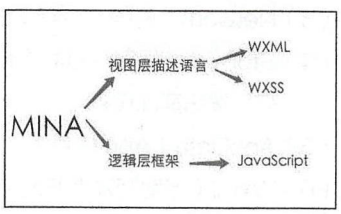


图 1-41 MINA 框架

MINA 的核心是一个响应的数据绑定系统。整个系统分为两块：视图层（View，描述语言 WXML 和 WXSS）和逻辑层（App Service，基于 JavaScript）。这可以让数据与视图非常简单地保持同步。当作数据修改的时候，只需要在逻辑层修改数据，视图层就会做出相应的更新。

项目创建好之后，包含了小程序的主体部分。该主体部分由 3 个文件组成，如表 1-7 所示，它们必须同时放在项目的根目录下，3 个代码文件是 app.js、app.json、app.wxss。其中，.js 后缀的是脚本文件，是小程序逻辑层；.json 后缀的文件是配置文件，负责小程序公共设置；.wxss 后缀的是小程序的公共样式表文件。微信小程序会读取这些文件，并生成小程序实例。

表 1-7　　小程序的主体部分

文件	项目必须	作用
app.js	是	小程序逻辑
app.json	是	小程序公共设置
app.wxss	否	小程序公共样式表

在 app.js 文件中可以监听并处理小程序的生命周期函数、声明全局变量、调用框架提供的丰富的 API，如本例的同步存储及同步读取本地数据。

在 app.json 文件中可以配置小程序由哪些页面组成，配置小程序的窗口、背景色，配置导航条样式，配置默认标题。注意该文件不可添加任何注释。

app.wxss 是整个小程序的公共样式表。我们可以在页面组件的 class 属性上直接使用。

pages 目录用于存储小程序的框架页面，一个页面是一个目录，如图 1-40 所示，目录下有 index（首页，欢迎页）和 logs（信息页，小程序启动日志的展示页）两个页面。单击首页“获取头像和昵称”，授权登录，然后单击头像会跳转到信息页，在信息页单击“返回”，可以退回到首页。微信小程序中每一个页面的【路径+页面名】都需要写在 app.json 的 pages 中，且 pages 中的第一个页面是小程序的首页。如图 1-42 所示。

app.json

```
{
  "pages":[
    "pages/index/index",
    "pages/logs/logs"
  ],
  "window":{
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#fff",
    "navigationBarTitleText": "WeChat",
    "navigationBarTextStyle":"black"
  }
}
```

图 1-42　app.json 的 pages 信息

一个框架页面是由同路径下同名的 4 个不同后缀的文件组成，如：index.js、index.wxml、index.wxss、index.json。.js 后缀的文件是脚本文件，.json 后缀的文件是配置文件，.wxss 后缀的文件是样式表文件，.wxml 后缀的文件是页面结构文件。如表 1-8 所示。

表 1-8　　框架页面组成部分

文件后缀	必填	作用
.js	是	页面逻辑
.wxml	是	页面结构
.wxss	否	页面样式表
.json	否	页面配置

● index.js 是页面的脚本文件，在这个文件中可以监听并处理页面的生命周期函数、获取小程序实例、声明并处理数据、响应页面交互事件等。代码如图 1-43 所示。

● index.wxml 是页面的结构文件，项目中使用了<view/>、<image/>、<text/>来搭建页面结构、绑定数据和交互处理函数。代码如图 1-44 所示。

index.js

```
//index.js
//获取应用实例
var app = getApp()
Page({
  data: {
    motto: 'Hello World',
    userInfo: {}
  },
  //事件处理函数
  bindViewTap: function() {
    wx.navigateTo({
      url: '../logs/logs'
    })
  },
  onLoad: function () {
    console.log('onLoad')
    var that = this
    //调用应用实例的方法获取全局数据
    app.getUserInfo(function(userInfo){
      //更新数据
      that.setData({
        userInfo:userInfo
      })
    })
  }
})

```

图 1-43　index.js 文件代码

index.wxml

```
<!--index.wxml-->
<view class="container">
  <view  bindtap="bindViewTap" class="userinfo">
    <image class="userinfo-avatar" src="{{userInfo.avatarUrl}}" background-size="cover"></image>
    <text class="userinfo-nickname">{{userInfo.nickName}}</text>
  </view>
  <view class="usermotto">
    <text class="user-motto">{{motto}}</text>
  </view>
</view>

```

图 1-44　index.wxml 文件代码

● index.wxss 是页面的样式表，页面的样式表不是必需的。当有页面样式表时，页面样式表中的样式规则会层叠覆盖 app.wxss 中的样式规则。如果不指定页面的样式表，也可以在页面的结构文件中直接使用 app.wxss 中指定的样式规则。代码如图 1-45 所示。

index.wxss

```
/**index.wxss**/
.userinfo {
  display: flex;
  flex-direction: column;
  align-items: center;
}

.userinfo-avatar {
  width: 128rpx;
  height: 128rpx;
  margin: 20rpx;
  border-radius: 50%;
}

.userinfo-nickname {
  color: #aaa;
}

.usermotto {
  margin-top: 200px;
}
```

图 1-45　index.wxss 文件代码

● index.json 是页面的配置文件，页面的配置文件不是必需的。当有页面配置文件时，配置项在该页面会覆盖 app.json 的 window 中相同的配置项。如果没有指定的页面配置文件，则在该页面直接使用 app.json 中的默认配置。

我们可以添加新的页面，也可以对现有页面进行重命名和删除操作。添加文件需要注意一个原则：为了方便开发者减少配置项，微信规定描述页面的这 4 个文件必须具有相同的路径与文件名。我们添加一个“circle”页面，如图 1-46 所示。

单击“circle.js”文件，右侧编辑区域可以书写代码，如图 1-47 所示。当然我们也可以编辑“circle.json”“circle.wxml”“circle.wxss”文件。

编辑器支持比较完善的自动补全功能，大大方便了开发者，代码编辑区域同时支持快捷操作，其快捷键功能与其他编辑器保持一致。

下面列出了一些常用的快捷键，Windows 平台下常用快捷键如表 1-9 所示。

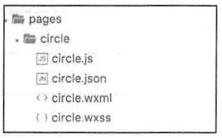


图 1-46 circle 页面

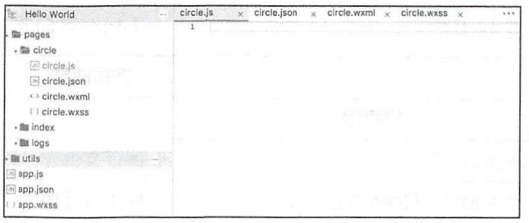


图 1-47 代码编辑区域

表 1-9 Windows 平台下常用快捷键

格式调整	
快捷键	**说明**
Ctrl+S	保存文件
Ctrl+[、Ctrl+]	代码行缩进
Ctrl+Shift+[Ctrl+Shift+]	折叠打开代码块
Ctrl+C、Ctrl+V	复制、粘贴，如果没有选中任何文字则复制、粘贴一行
Shift+Alt+F	代码格式化
Alt+Up、Alt+Down	上下移动一行
Shift+Alt+Up、Shift+Alt+Down	向上向下复制一行
Ctrl+Shift+Enter	在当前行上方插入一行
与光标相关	
快捷键	**说明**
Ctrl+End	移动到文件结尾
Ctrl+Home	移动到文件开头
Ctrl+I	选中当前行
Shift+End	选择从光标到行尾
Shift+Home	选择从行首到光标处
Ctrl+Shift+L	选中所有匹配
Ctrl+D	选中匹配
Ctrl+U	光标回退
与界面相关	
快捷键	**说明**
Ctrl+\	打开或隐藏目录结构
Ctrl+M	打开或隐藏页面实时展示区（模拟器）
Ctrl+N	新建临时文件
Ctrl+PageUp	跳转到左边文件
Ctrl+PageDown	跳转到右边文件

Mac 平台下常用快捷键如表 1-10 所示。

表 1-10　　Mac 平台常用快捷键

格式调整	
快捷键	**说明**
Command+S	保存文件
Command+[、Command+]	代码行缩进
Command+Option+[Command+Option+]	折叠打开代码块
Command+C、Command+V	复制、粘贴，如果没有选中任何文字则复制、粘贴一行
Option+Shift+F	代码格式化
Option+↑、Option+↓	上下移动一行
Option+Shift+↑、Option+Shift+↓	向上向下复制一行
Command+Shift+Enter	在当前行上方插入一行
与光标相关	
快捷键	**说明**
Command+→	移动到文件结尾
Command+←	移动到文件开头
Command+l	选中当前行
Command+Shift+→	选择从光标到行尾
Command+Shift+←	选择从行首到光标处
Command+Shift+↑	选择从光标到上一行
Command+Shift+↓	选择从光标到下一行
Command+Option+L	选中所有匹配
Command+D	选中匹配
Command+U	光标回退
与界面相关	
快捷键	**说明**
Command+\	打开或隐藏目录结构
Option+M	打开或隐藏页面实时展示区（模拟器）
Command+N	新建临时文件
Command+PageUp	跳转到左边文件
Command+PageDown	跳转到右边文件

除了上面常用的快捷键，用户还可以通过导航菜单区域中的“调试器”-更多按钮“⋮”选择“Shortcuts”选项，查看所有的快捷方式，如图 1-48 所示。

所有快捷键，如图 1-49 所示。

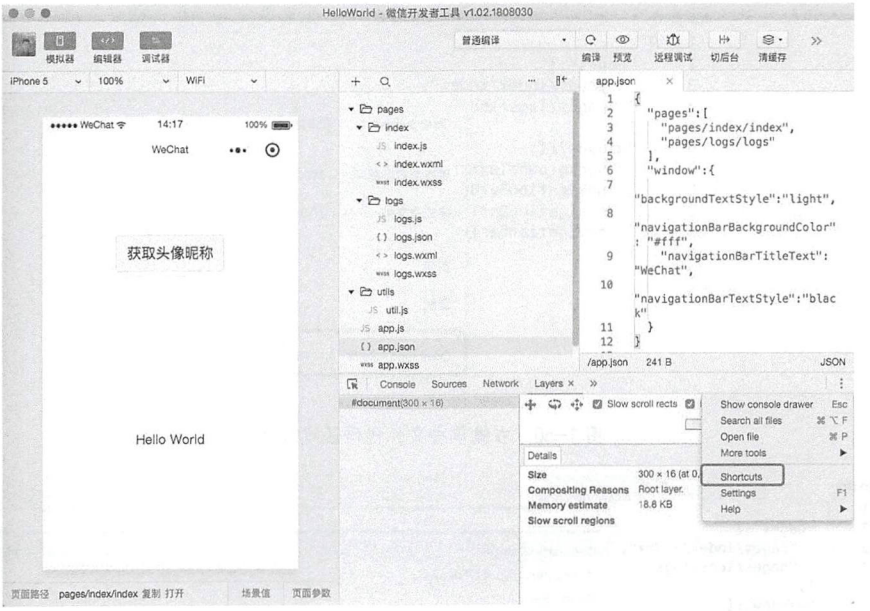


图 1-48 快捷键入口

图 1-49 所有快捷键

也可以打开任意文件，然后右键单击代码区域，如图 1-50 所示。

单击“Command Palette”即可查看所有命令选项板，如图 1-51 所示。

utils 目录主要存放工具脚本，项目自带了一个 util.js 脚本文件，里面的 formatTime 方法主要用于计算日期，代码如图 1-52 所示。

图 1-50 右键单击文件代码区域效果

图 1-51 所有命令选项板

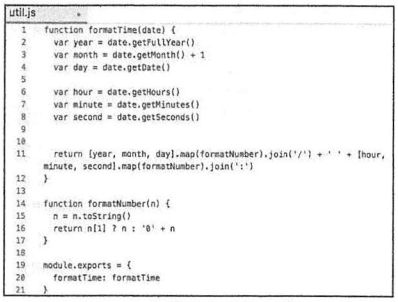


图 1-52 util.js 脚本文件

1.4.3 调试器

导航菜单区域中的“调试器”功能，帮助开发者检测代码的实现与排查问题。

调试器常用功能分为 8 页功能模块：Console、Sources、Network、Storage、AppData、Wxml、Sensor、Trace 和最右侧更多按钮“⋮”，下面将逐个介绍每一页功能。

1）Console 页

Console（控制台信息）页，这里可以查看小程序的错误 log 信息输出，如图 1-53 所示。

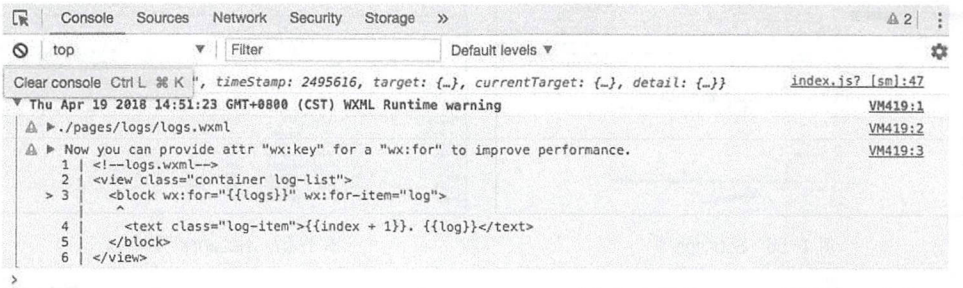


图 1-53 错误信息

开发者也可以在此直接输入代码并调试，如图 1-54 所示。我们先写了一个 formatTime 方法，书写正确，没有任何错误，代码结束按下回车键之后可以看到信息栏输出“<.undefined”表示没有发现错误。第二次我们在获取年的最后多写了一个“1”，代码结束按下回车键之后可以看到信息栏输出“Uncaught SyntaxError: Unexpected number”，后面显示“VM307.2”，双击“VM307.2”之后，可以定位到错误位置，如图 1-55 所示。

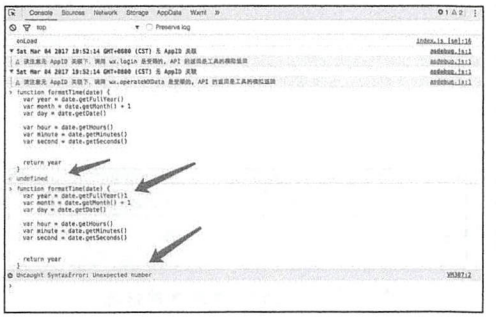


图 1-54 Console 页调试

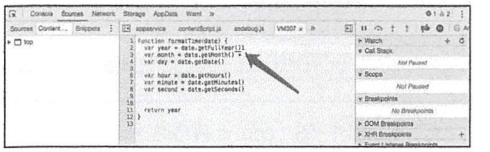


图 1-55 错误定位

2）Sources 页

Sources（源文件）页，用于显示当前项目的脚本文件，如图 1-56 所示。同浏览器开发不同，微信小程序框架会对脚本文件进行编译的工作，所以在源文件页中，我们看到的文件其实是经过处理之后的脚本文件，我们编写的代码都会被包裹在 define()函数中。并且对于页面（Page）的代码，在尾部会有 require 的主动调用。

3）Network 页

Network（网络）页，显示网络相关的详细信息，用于观察和显示每个元素的请求（request）信息和套接字（socket）的请求情况，如图 1-57 所示。

图 1-56 Sources 页

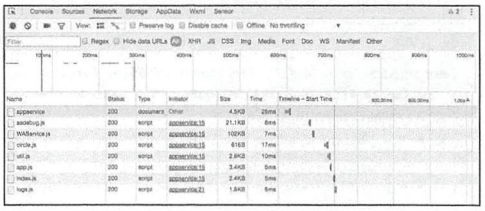


图 1-57 Network 页

单击一个文件，可以查看该文件的 Headers、Preview、Response、Timing 等信息，如图 1-58 所示。

4）Storage 页

Storage（数据存储）页，在项目中使用 wx.setStorage 或者 wx.setStorageSync 存储 API 时，会缓存一些数据信息，这些在 Storage 页里面可以看到。例如我们在控制台信息页（Console）中输入：wx.setStorage({key:"name",data:"LiuMingyang"})，在 Storage 页里就可以找到对应的 Key-Value 数据，如图 1-59 所示。

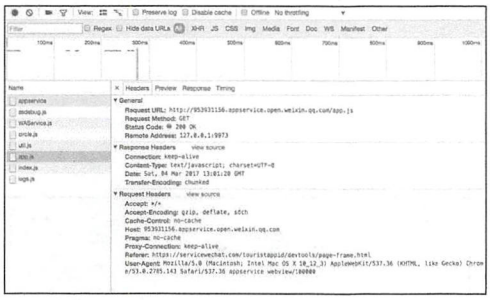


图 1-58 app.js 网络信息

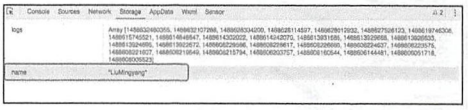


图 1-59 Storage 页缓存的信息

5）AppData 页

AppData 页用于显示当前小程序项目当前时刻使用的具体数据，并且实时地反馈项目数据情况。我们可以在此处编辑数据，数据将会及时地反馈到界面上。如图 1-60 所示。

6）Wxml 页

Wxml 页用于帮助开发者开发 Wxml 转化后的界面。在这里可以看到真实的页面结构以及结构对应的 wxss 属性，同时我们可以通过修改对应的 wxss 属性，在模拟器中实时看到修

改的情况。通过调试区域左上角的选择器“”，我们还可以快速找到页面中组件对应的 Wxml 代码。

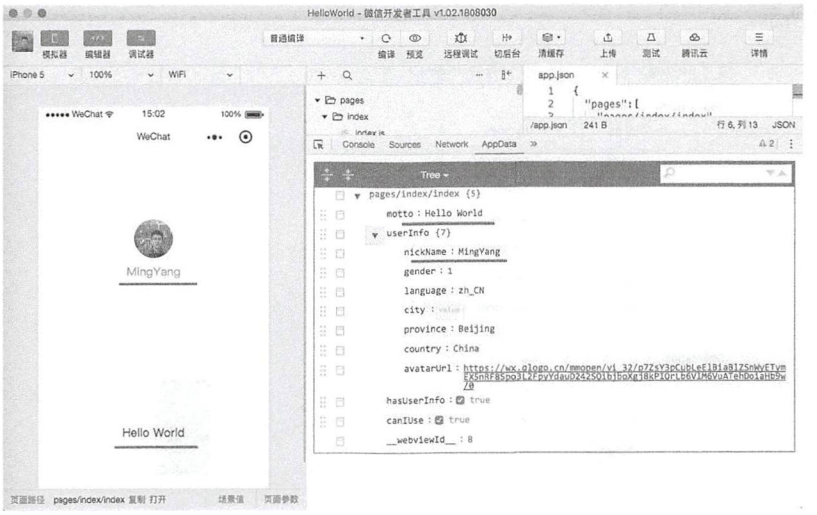


图 1-60 AppData 页

单击左上角的选择器“”，按钮会变成蓝色“”。在模拟器中，开发者可以定位选择某一个控件，例如选择头像后，选择器会自动定位到 Wxml 中对应的代码段，最右侧会显示对应的样式，如图 1-61 所示。这样方便调整样式或 Wxml 中的代码，调整后立即可以看到效果。这里需要注意，每次定位选择之前，必须单击左上角的选择器“”，使其变成蓝色“”。

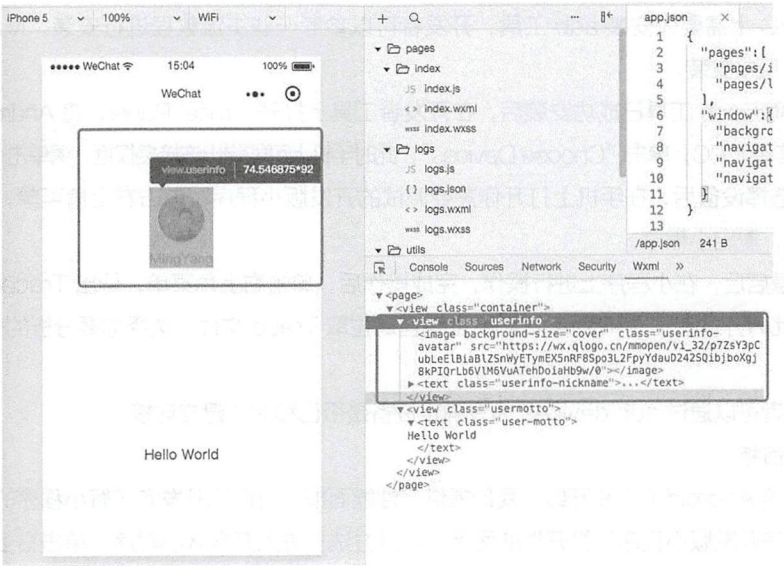


图 1-61 Wxml 页

7）Sensor 页

Sensor（传感器）页有两大功能：模拟地理位置或移动设备表现，以及调试重力感应。如图 1-62 所示。

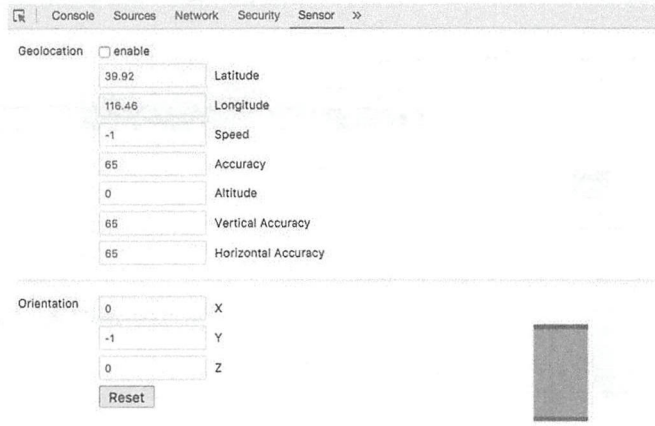


图 1-62 Sensor 页

8）Trace 页

从微信 Android 6.5.10 开始支持，提供了 Trace 导出工具，开发者可以在开发者工具 Trace Panel 中使用该功能，如图 1-63 所示。

使用方法

1. PC 上需要先安装 adb 工具，开发者可以参考一些主流教程进行安装，Mac 上可使用 brew 直接安装。

2. 确定 adb 工具已成功安装后，在开发者工具上打开 Trace Panel，将 Android 手机通过 USB 连接上 PC，单击“Choose Devices”，此时手机上可能弹出连接授权框，请单击“允许”。

3. 选择设备后，在手机上打开你需要调试的开发版小程序，单击右上角菜单，打开性能监控面板，重启小程序。

4. 重启后，在小程序上进行操作，完成操作后，单击右上角菜单，导出 Trace 数据。

5. 此时开发者工具 Trace Panel 上会自动拉取 Trace 文件，选择你要分析的 Trace 文件即可。

开发者可以通过 adb devices 命令确定设备是否已和 PC 建立连接。

性能面板

从微信 Android 6.5.8 开始，我们提供了性能面板，可以让开发者了解小程序的性能。开发者可以在开发版小程序下打开性能面板，打开方法：进入开发版小程序，单击右上角更多按钮，再单击“显示性能窗口”，如图 1-64 所示。

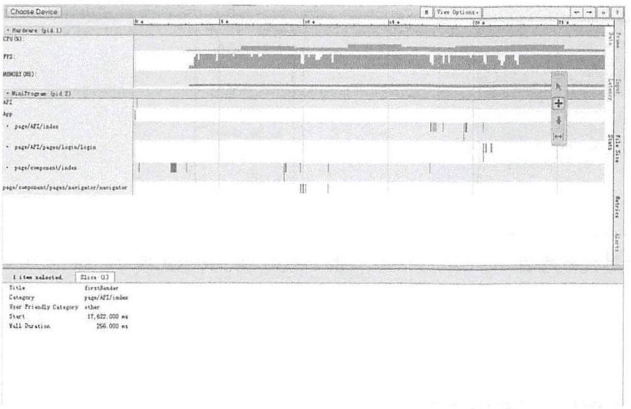


图 1-63 Trace 功能演示

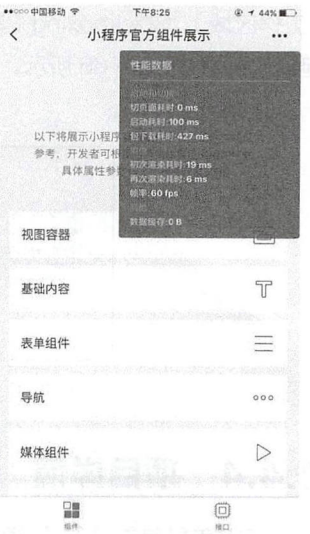


图 1-64 性能面板

性能面板指标说明，如表 1-11 所示。

表 1-11 性能面板指标说明

指标	说明
CPU	小程序进程的 CPU 占用率，仅 Android 下提供
内存	小程序进程的内存占用（Total Pss），仅 Android 下提供
启动耗时	小程序启动总耗时
下载耗时	小程序包下载耗时，首次打开或资源包需更新时会进行下载
页面切换耗时	小程序页面切换的耗时
帧率/FPS	每秒显示帧数
首次渲染耗时	页面首次渲染的耗时
再次渲染耗时	页面再次渲染的耗时（通常由开发者的 setData 操作触发）
数据缓存	小程序通过 Storage 接口储存的缓存大小

9）最右侧更多按钮“⋮”

扩展菜单选项，主要包括开发工具的一些定制与设置，如图 1-65 所示。

“Show console”：显示控制台页。

“Search all files”：查找文件。

“More tools”：更多工具，包括 Inspect devices（检测设备）、Network conditions（网络条件）、Rendering settings（渲染设置）、Sensors（重力传感器）。

“Shortcuts”：快捷键自定义或配置。

“Settings”：开发者工具的环境参数设定，包括 Preferences（喜好）、Workspace

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

（工作区域）、Blackboxing（黑箱）、Devices（模拟器手机类型）、Throttling（网络带宽及延时限制），如图 1-66 所示。

“Help”：帮助。

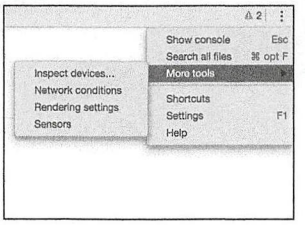


图 1-65 扩展菜单选项

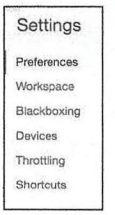


图 1-66 Settings选项

1.4.4 项目详情

在菜单栏单击“工具-项目详情”，或者单击开发工具顶部右侧“详情”按钮，打开项目详情，主要有 4 部分功能，当前项目细节、项目设置、域名信息、腾讯云状态。如图 1-67 所示。

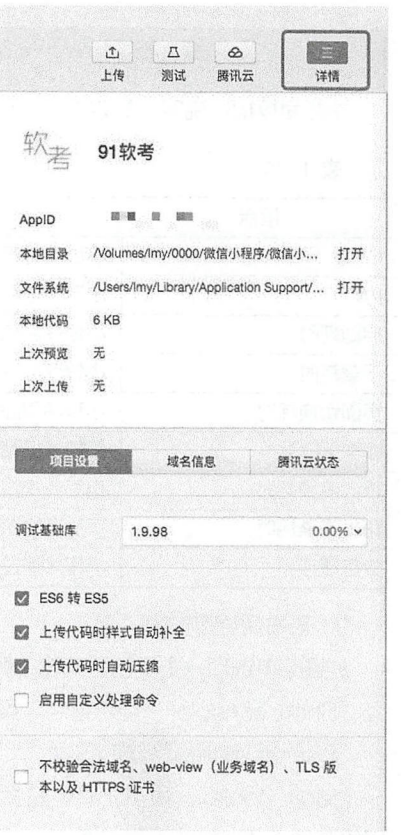


图 1-67 项目详情

1）当前项目细节

该功能包括图标、AppID、目录信息，以及上次提交代码的时间和代码包大小。

2）项目设置

ES6 转 ES5

微信小程序运行在三端：iOS、Android 和用于调试的开发者工具，三端的脚本执行环境以及用于渲染非原生组件的环境是各不相同的：

- 在 iOS 上，小程序的 JavaScript 代码是运行在 JavaScriptCore 中，是由 WKWebView 来渲染的，环境有 iOS8、iOS9、iOS10。
- 在 Android 上，小程序的 JavaScript 代码是通过 X5 JSCore 来解析，是由 X5 基于 Mobile Chrome 37 内核来渲染的。
- 在开发工具上，小程序的 JavaScript 代码是运行在 nwjs 中，是由 Chrome Webview 来渲染的。

虽然三端的环境是十分相似的，但是至少目前还

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

是有一些区别的：

● ES6 语法支持不一致，语法上开发者可以通过开启 ES6 转 ES5 的功能来规避。微信在 0.10.101000 以及之后版本的开发工具中，会默认使用 babel 将开发者代码 ES6 语法转换为三端都能很好支持的 ES5 代码，帮助开发者解决不同环境所带来的开发问题。开发者可以在项目设置中关闭这个功能。

● ES6 API 支持不一致，考虑到代码包大小的限制，API 上目前需要开发者自行引入相关的类库来进行处理，例如 Promise 需要开发者自行引入 Polyfill 或者别的类库。开发者可以通过 caniuse 或者 X5 兼容查询到相关 API 的支持情况。

● wxss 渲染表现不一致，尽管可以通过开启上传代码时样式自动补全来规避大部分的问题，我们还是建议开发者在 iOS 和 Android 上检查小程序的真实表现。

上传代码时样式自动补全

开启此选项，开发工具会自动检测并补全缺失样式，保证在 iOS8 上的正常显示。

上传代码时自动压缩

开启此选项，开发工具在上传代码时将会帮助开发者压缩 JavaScript 代码，减小代码包体积。

启用自定义处理命令

开启此选项，与当前项目相关的文件发生改变时，开发工具会自动刷新调试模拟器，从而提高开发效率。

不校验合法域名、web-view（业务域名）、TLS 版本以及 HTTPS 证书

开启此选项，开发工具将不会校验合法域名以及 TLS 版本等，帮助开发者在开发过程中更好地完成调试工作。

3）域名信息

该功能显示后台配置的合法域名。

4）腾讯云状态

该功能显示是否授权腾讯云等信息。

1.4.5 运行预览

开启了“启用自定义处理命令”，在编辑代码文件的时候，页面实时展示区域（模拟器）会自动刷新。开发者也可以通过单击左侧导航菜单区域中的“调试器”功能，在页面实时展示区域（模拟器）中运行小程序，查看小程序运行结果。

作为开发者，单击工具栏“预览”工具会自动上传源代码到微信服务器，成功之后显示一个二维码，开发者用微信扫描二维码即可看到相应项目，如图 1-68 所示。

为了方便测试某一个界面，开发者可以自定义预览，编译模式选择“添加编译模式”，如

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

图 1-69 所示。

图 1-68 安装二维码

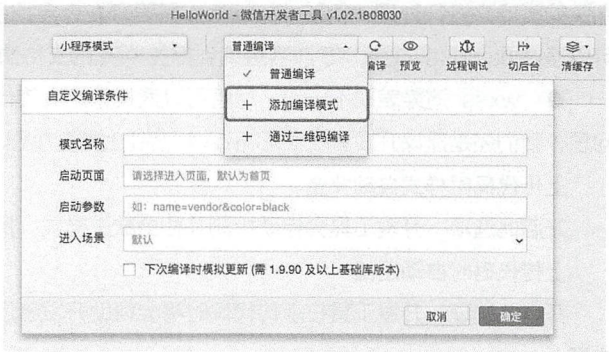


图 1-69 自定义预览功能

单击开发工具顶部的“上传”按钮，开发工具会上传源代码到微信服务器，开发者可以在微信公共平台管理后台看到提交的情况。需要注意的是，代码上传功能只能用管理员微信号操作。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

第二章

小程序详细介绍

本章主要带你对比小程序与原生 App 应用和 HTML5 应用的区别，进一步了解小程序的应用场景。本章从开发基础入手，通过“HelloWorld”小程序项目，详细介绍小程序开发所需要的基础知识：配置信息、MINA 框架结构、生命周期、数据绑定、条件渲染、列表渲染、模板、事件、引用、样式展现等信息，以及小程序的开发步骤、上传和发布流程。

2.1 小程序、原生 App、WebApp 的区别

➢ 小程序

小程序是一种用户只需要扫描二维码或搜一搜即可打开的应用，无须下载安装即可使用的手机“应用”。开发是基于统一框架进行的，框架提供了标准界面模板，通过提供本地的 API 供 H5 上的 JS 调用，但运行比 H5 更顺畅。

下面列出小程序明显的几点优缺点。

小程序的优点：

1. 无须下载安装，无须注册，即开即用，用完就走，不占用手机内存，省流量，省安装时间；
2. 打开速度比 H5 还快，体验上接近原生 App；
3. 跨越安卓和苹果平台，开发成本比 App 低，可以将更多财力、人力、精力放在产品运营和内容本身；
4. 安卓手机可以直接添加手机桌面，看上去和 App 差不多；
5. 相较于各种 App，微信小程序 UI 和操作流程会更统一，降低了用户的使用难度；
6. 相较于原生 App，微信小程序推广更简单，更省成本。

小程序的缺点：

1. 只能分享给群、朋友，不能分享到朋友圈；
2. 二维码长按识别之后不能直接进入小程序，只能通过微信扫一扫才能进入到小程序；
3. 没有推送（push）功能，不能给用户推送消息；

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

4. 所能获取的用户数据非常有限，基本就是头像、昵称等，没有用户体系；

5. 不需要注册，用完即走，没办法多任务处理；

6. 不能做游戏；

7. 小而美，做垂直，功能复杂度有限制。

➢ 原生 App

原生 App 也称为 Native App，基于不同智能手机操作系统，如 iOS、Android、Windows，采用不同的语言和框架进行开发，该模式通常是由“服务器+App 客户端”两部分构成，App 应用所有的 UI 元素、数据内容、逻辑框架均安装在手机终端上，才可以使用。

原生 App 与小程序几点对比：

1. 小程序开发比原生 App 更简单，开发周期更短。小程序提供框架和 API，基于 HTML5 进行开发，对接开发者现有的 App 后台的用户数据，其开发难度相对 App 较低；

2. 小程序开发成本比原生 App 更低。原生 App 开发需要的人力、物力和时间成本都比较高；

3. 小程序可以满足一些简单的基础应用，低频次及偏向于线下和场景生活服务类的轻应用，在餐饮、快递等行业其受欢迎程度高。而对于一些需要大量计算的功能类应用，如图片处理或文档编辑，小程序是无法满足的，只能用原生 App 或者 WebApp 去实现；

4. 原生 App 的 UI、UE 可以设计得更加人性化、更绚丽，功能的完善完全取决于开发者的想象力和技术实力。对系统接口的调用更为简单，一些功能比如 AR（增强现实技术）、语音识别等，App 能够在交互、视觉等用户体验上满足用户的高要求。对于更丰富、更细化、更个性化的功能，是需要更大容量实现的，这就需要在 App 上去承载，而小程序作为轻量级应用是无法满足的；

5. 原生 App 应用可以在 App Store、Android 市场、360 手机助手、百度应用、安全管家等应用市场进行下载安装。微信小程序只能通过扫描二维码或搜索小程序的名称，以及微信群或好友分享获取；另外，如果微信公众号关联了小程序，那么在公众号的介绍页面会出现一个相关小程序的模块。只要单击相应的图标，就能轻松打开；

6. 安装流程，小程序更简单。打开微信扫描二维码即可进入小程序。App 需要扫码或者搜索进行下载并安装后才可以使用；

7. 原生 App 安装在手机上会占用一定的空间，且随着功能的不断完善和更新其占用的资源也会更多。而微信小程序无须下载，其内容和功能都不占用手机内存；但微信本身比较臃肿，好友会话、群消息、朋友圈等功能堆积会占用大量的手机内存；

8. 功能扩展性，App 更强；

9. App 的维护成本较高，需要针对不同操作系统做兼容性的开发，且需要用户自行升级。小程序运行于微信平台，大部分的维护工作由腾讯完成，其维护成本低、周期短、流程简单，更新也主要在微信后端完成，不存在操作系统和浏览器兼容方面的问题；

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

10. App 用户忠诚度更高，因为 App 的内容全面，用户主动选择，并且 App 获取门槛高，二次消费门槛很低。小程序的内容碎片化，用户被动接受，用完就退出程序，虽然首次消费门槛低，但后续消费门槛无法降低，用户由于难以找到小程序而选择放弃；

11. 小程序只是简化版的 App，并不能涵盖 App 的全部内容。微信对小程序的内存大小限制在 2MB，只能保存最基础的功能。App 更适合于高频场景的服务，天然具备独立发展成生态的潜力，可以自成一套生态体系。即使小程序在用户体验上大大增强，这些也只是增量，App 还是主要的阵地；

12. 小程序由于微信本身的传播能力和获客能力，可以让互联网创业公司减少试错成本，提高产品的成功率。但微信缺少小程序的展示位，小程序难以获得好的位置资源，对于初创品牌来说如何让用户搜索到并引导用户单击前往是一大难题。

➢ WebApp

WebApp 也就是我们说的 HTML5 App，是一种框架型 App 开发模式。具有跨平台的优势，该开发模式通常由“HTML5 云网站+App 应用客户端”两部分构成，App 应用客户端只需安装应用的框架部分，应用所有界面都是通过 HTML5 实现的，每次打开 App 的时候，都是去云端获取数据呈现给手机用户。

WebApp 与小程序几点对比：

1. 运行环境的不同。HTML5 的运行环境是浏览器，包括 webview，而微信小程序的运行环境并非完整的浏览器。小程序在开发过程中会用到 HTML5 相关的技术（并非全部），最后的发布上线需要微信审核，微信在不更新自身软件的情况下可以将小程序更新到自身软件内，这就联想到了 React Native 框架，并且已经有开发者在微信小程序的开发工具源码中发现使用了 React 和 NodeWebkit 库。小程序的运行环境很有可能是微信开发团队基于浏览器内核完全重构的一个内置解析器，针对小程序专门做了优化，配合自己定义的开发语言标准，提升了小程序的性能；

2. 开发成本不同。微信小程序的开发直接使用微信团队提供的开发工具，并规范了开发标准，将前端常见的 HTML、CSS 变成了自定义的 WXML、WXSS，WXML 标签中、官方文档中也有明确的使用介绍，相信上手应该是非常容易的；WXSS、JSON 和 JS 文件中的写法稍有限制，但整体相差不多。在统一了这些标准之后，作为一个开发者，你会发现，自己只要专注写程序就可以了。并且在使用一些 API 时，不用再去顾虑浏览器兼容性，不用担心生产环境中出现不可预料的奇妙 BUG。对一个 HTML5 web 开发者，除了要熟悉复杂的开发工具（vscode、sublimtext、Atom 等），还要学习前端框架（Angular、react、vue、backbone 等）、模块管理工具（Webpack、Browserify 等）、任务管理工具（Grunt、Gulp 等），小到 UI 库选择、接口调用工具（ajax、Fetch Api 等）、浏览器兼容性等都要我们一一考虑。尽管这些工具可定制化非常高，并且提高了开发者的开发效率，但我相信项目开发的配置工作已经消耗了不少精力。项目中使用的各种外部库、各种工具的版本迭代、版本升级所产生

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

的成本应该也不低。可见微信小程序的开发成本相比以往的 web 开发确实低很多；

3. 获取系统级权限的不同。微信小程序相对于 HTML5 web 应用能获得更多的系统权限，比如网络通信状态、数据缓存能力等，这些系统级权限都可以和微信小程序无缝衔接，也就是官方宣称的拥有 Native App 的流畅性能，而这一点恰巧是 HTML5 web 应用经常被诟病的地方，也是 HTML5 的大多应用场景被定位在业务逻辑简单、功能单一的原因；

4. 运行流畅度的不同。无论对于用户还是开发者来说，这点都是最直观的感受。HTML5 应用面对复杂的业务逻辑或者丰富的页面交互时，体验总是不尽人意，需要不断地对项目进行优化来提升用户体验。由于微信小程序运行环境独立，尽管同样用 html+CSS+js 去开发，但配合微信的解析器最终渲染出来的是原生组件的效果，自然体验上更流畅。

2.2 应用场景

在了解其功能和特点后，那么，小程序适用于什么应用场景呢？

根据上一节不同应用之间的对比，我们可以总结出以下几点：

1. 小程序不适用于高频、刚需的应用场景，该场景下应采用原生 App，不适合采用小程序；

2. 小程序作为增量渠道，为原生 App 进行导流。对于高频、非刚需的应用场景，采用的产品形态视情况而定，小程序主要适用于：

- 偏工具的内容型产品。
- 日常工具类产品。
- 社区类产品（作为导流作用）。
- 创业者进行 MVP 产品形式的探索。

3. 初创企业进行产品模式的探索；

4. 对于低频、非刚需需求，基本是属于小众的需求，一般有两种情况：开发者自身兴趣；专业级产品，面向某领域专业用户。

- 对于第一种情况，就看开发者本身的能力，如果你是移动端开发者，那就开发原生 App；如果你是前端开发者，就开发小程序。
- 对于第二种情况，由于专业级产品一般对于性能和交互体验较高，所以优先采用原生 App。

5. 基本涵盖所有低频、刚需的长尾生活服务需求场景。

任何一种线上的场景，如果用户体验很差，自然就无法继续流行下去，客户的行为也会影响商家，所以比较大的商家更倾向于使用保证用户体验的本地网络+App 的解决方案。

小程序由于在本地执行，体验会非常流畅。例如，点餐入口，都是使用贴在桌面上的二维码，用户需要时，可以随时呼出小程序。所以，点餐是小程序一个非常有代表性的应用场景。张小龙在演讲时举的第一个例子，就是微信小程序点餐的场景。如果在整个场景中，用户需要完成多个流程，并且流程之间的流畅性对客户体验影响很大，例如

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

购电影票选位支付、买机票值机、买汽车票支付等，那么这一类就是微信小程序的场景化应用。

2.3 全局配置（app.json）和页面配置（*.json）

2.3.1 全局配置 app.json 详解

微信小工具通过使用 app.json 文件来进行全局配置，可以配置页面文件的路径（“pages”）、窗口的表现效果（“window”）、设置网络超时时间值（“networkTimeout”）、配置多页面切换（“tabBar”）、是否开启 debug 模式（“debug”）。app.json 为标准的 json 文件，不能存在注释，所以使用的时候需把注释全部去掉。

app.json 文件代码内容，如图 2-1 所示。

app.json 文件中的全局配置并不多，表 2-1 列出了 5 个配置项，每个配置项的数据类型并不相同，也并非都是必须设置。通过表 2-1 全局配置项的描述，开发者可以更好地了解全局配置。

app.json

```
{
  "pages":[
    "pages/index/index",
    "pages/found/found",
    "pages/logs/logs"
  ],
  "window":{
    "navigationBarBackgroundColor": "#ff5400",
    "navigationBarTextStyle":"black",
    "navigationBarTitleText": "Second Chapter",
    "backgroundColor": "#999999",
    "backgroundTextStyle":"light",
    "enablePullDownRefresh": true
  },
  "tabBar": {
    "list": [
      {
        "pagePath": "pages/index/index",
        "text": "首页"
      },
      {
        "pagePath": "pages/logs/logs",
        "text": "日志"
      }
    ]
  },
  "networkTimeout": {
    "request": 10000,
    "downloadFile": 10000
  },
    "debug": true
}
```

图 2-1 app.json 文件代码内容

表 2-1 全局配置项

配置项	类型	必填	描述
pages	Array	是	设置页面路径
window	Object	否	设置默认页面的窗口表现
tabBar	Object	否	设置底部 tab 的表现
networkTimeout	Object	否	设置网络超时时间
debug	Boolean	否	设置是否开启 debug 模式

1）pages 配置项

pages（Array）必填，用于设置页面路径。接收一个数组，每一项都是字符串，小程序所有页面都需要写在 pages 数组里面，来指定小程序由哪些页面组成。每一项代表对应页面的“路径+文件名”信息。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

pages 配置项遵循以下 3 个原则：

（1）数组的第一项代表小程序的初始页面；

（2）小程序中新增/减少页面，都需要对 pages 数组进行修改；

（3）文件名不需要写文件后缀，因为框架会自动去寻找路径.json、.js、.wxml、.wxss 的 4 个文件进行整合。

如开发目录为：

```
pages/
pages/ found/ found.wxml
pages/ found/ found.js
pages/index/index.wxml
pages/index/index.js
pages/index/index.wxss
pages/logs/logs.wxml
pages/logs/logs.js
app.js
app.json
app.wxss
```

则在 app.json 中 pages 的配置应如下所示：

```
{
  "pages":[
    "pages/index/index",
    "pages/found/found",
    "pages/logs/logs"
  ],
}
```

2）window 配置项

window（Object）非必填配置项，没有该配置项时将使用系统默认配置。window 配置项用来设置默认界面的窗口表现，用于设置小程序的状态栏、导航条、标题、窗口等对象的颜色、背景颜色、内容等属性。

window 配置项的描述如表 2-2 所示。

表 2-2　　window 配置项

对象	类型	默认值	描述
navigationBarBackgroundColor	HexColor	#000000	导航栏背景颜色，十六进制颜色类型
navigationBarTextStyle	String	white	导航栏标题颜色，仅支持 black、white
navigationBarTitleText	String		导航栏标题内容
backgroundColor	HexColor	#ffffff	窗口背景颜色，在上拉刷新、下拉刷新、navigate 切换页面时可以看见
backgroundTextStyle	String	dark	下拉背景字体、loading 图的样式，仅支持 dark、light（String 形式）
enablePullDownRefresh	Boolean	false	是否开启下拉刷新

非卖品，仅供非商业用途或交流学习使用

如 app.json 中 window 配置如下：

```
{
  "window":{
    "navigationBarBackgroundColor": "#ff5400",
    "navigationBarTextStyle":"black",
    "navigationBarTitleText": "Second Chapter",
    "backgroundColor": "#999999",
    "backgroundTextStyle":"light",
    "enablePullDownRefresh": true
  }
}
```

则小程序的界面效果如图 2-2 所示。

图 2-2 配置 window 后界面效果

3）tabBar 配置项

tabBar（Object）设置底部 tab 的表现。一般原生 App 底部都会有工具栏，可进行页面切换。小程序可以通过 tabBar 配置项指定 tab 栏的表现，以及 tab 切换时显示的对应页面，实现多页面切换。tabBar 内部是一个“list”数组，只能配置最少 2 个、最多 5 个 tab，tab 按数组的顺序排序。

具体示例代码如下所示：

```
{
 "tabBar": {
    "list": [
      {
        "pagePath": "pages/index/index",
        "text": "首页"
      },
      {
        "pagePath": "pages/logs/logs",
        "text": "日志"
      }
    ]
  }
}
```

tabBar 内除了必需的“list”数组之外，我们还可以对其设置一些属性，如表 2-3 所示。

表 2-3 tabBar 相关属性配置

对象	类型	必填	默认值	描述
color	HexColor	是		tab 上的文字默认颜色
selectedColor	HexColor	是		tab 上的文字选中时的颜色
backgroundColor	HexColor	是		tab 的背景颜色
BorderStyle	String	否	black	tabbar 上边框的颜色，仅支持 black/white
list	Array	是		tab 的列表，最少 2 个、最多 5 个 tab
position	String	否	bottom	tabBar 位置，可选值为顶部（top）、底部（bottom）

其中 list 是一个数组，用于配置标签页，最少 2 个、最多 5 个标签页，标签页按数组的顺序排序。数组中的每个项都是一个对象，其属性值如表 2-4 所示。

表 2-4　　list 中对象相关属性

对象	类型	必填	描述
pagePath	String	是	页面路径，必须在 pages 中先定义
text	String	是	tab 上按钮文字
iconPath	String	是	tab 上 icon 图片路径，icon 大小限制为 40KB
selectedIconPath	String	是	选中 tab 时的图片路径，icon 大小限制为 40KB

如 app.json 中 tabBar 配置如下：

```
{
  "tabBar": {
    "color":"#6fdc56",
    "selectedColor":"#999999",
    "backgroundColor":"#ffffff",
    "borderStyle":"black",
    "position":"bottom",
     "list": [
       {
         "pagePath": "pages/index/index",
         "text": "首页",
         "iconPath":"images/icon_1.png",
         "selectedIconPath":"images/icon_1H.png"
       },
       {
         "pagePath": "pages/logs/logs",
         "text": "日志",
         "iconPath":"images/icon_2.png",
         "selectedIconPath":"images/icon_2H.png"
       }
     ]
   }
}
```

则小程序的界面效果如图 2-3 所示。

图 2-3　配置 tabBar 后界面效果

4）networkTimeout 配置项

networkTimeout（Object）设置各种网络请求的超时时间，非必须配置项。可设置的 networkTimeout 相关属性如表 2-5 所示。所有配置项单位均是 ms，如果超时不设置，则默认使用操作系统内核或遵循服务器 WebServer 的设定值。

表 2-5 networkTimeout 配置项

属性	类型	必填	说明
request	Number	否	wx.request 的超时时间，单位为 ms
connectSocket	Number	否	wx.connectSocket 的超时时间，单位为 ms
uploadFile	Number	否	wx.uploadFile 的超时时间，单位为 ms
downloadFile	Number	否	wx.downloadFile 的超时时间，单位为 ms

为了提供网络效率，可以在 app.json 中使用下面的超时设置：

```
{
 "networkTimeout": {
    "request": 20000,
    "connectSocket":2000,
    "uploadFile":2000,
    "downloadFile": 20000
  }
}
```

5）debug 配置项

debug（Boolean）设置是否开启 debug 模式，默认值是 false（关闭）状态。开发者可以在开发者工具中开启 debug 模式。在开发者工具的控制台面板，调试信息以 info 的形式给出，其信息有 Page 的注册、页面路由、数据更新、事件触发，可以帮助开发者快速定位一些常见的问题。

以下配置是打开 debug（调试）模式：

```
{
   "debug": true
}
```

开启“调试”模式之后，重启小程序，在 Console 页面可以看到更多的信息，如图 2-4 所示。

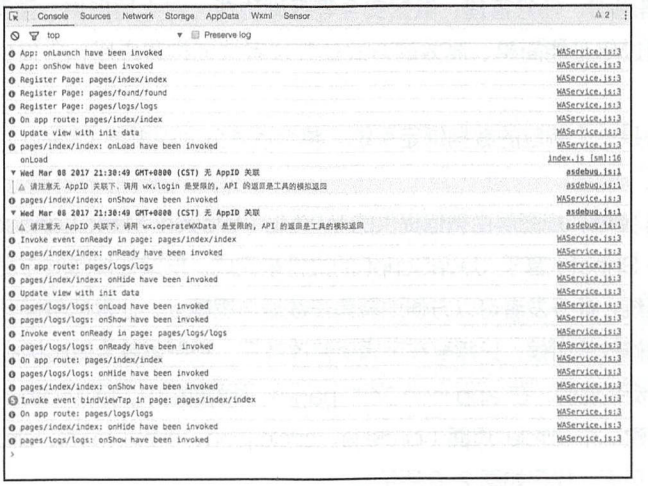


图 2-4 开启 debug 模式后 Console 页面输出的信息

利用 Console 页面的信息开发者可以快速定位一些常见的问题，需要注意的是，在提交审核发布时需要关闭 debug 模式。

2.3.2 页面配置（*.json）

每一个小程序页面也可以使用 .json 文件来对本页面的窗口表现进行配置。页面的配置比 app.json 全局配置简单得多，只是设置 app.json 中的 window 配置项的内容，页面中的配置项会覆盖 app.json 的 window 中相同的配置项。

页面的 .json 只能设置 window 相关的配置项，来决定本页面的窗口表现，所以无须再写 window 这个键，示例代码如下所示：

```
{
"navigationBarBackgroundColor": "#ffffff",
"navigationBarTextStyle": "black",
"navigationBarTitleText": "查看启动日志",
"backgroundColor": "#ff0589",
"backgroundTextStyle": "dark"
}
```

2.4 小程序架构

2.4.1 框架介绍

小程序开发使用的框架叫 MINA 框架，如图 2-5 所示，通过封装微信客户端提供的文件系统、网络通信、任务管理、数据安全等基础功能，对上层提供一整套的 JavaScript API，目标是通过尽可能简单、高效的方式让开发者快速构建应用，并提供具有原生 App 体验的服务。

框架的核心是一个响应的数据绑定系统。整个系统分为视图层（View）和逻辑层（App Service）两块。

逻辑层由 js 完成，视图层由微信提供的 WXML（WeiXin Mark Language）和 WXSS（WeiXin Style Sheet）基于 JavaScript 的文件来完成。框架在视图层与逻辑层间提供了数据传输和事件系统，让开发者可以方便地聚焦于数据与逻辑上，让数据与视图非常简单地保持同步。当作数据修改的时候，只需要在逻辑层修改数据，视图层就会做出相应的更新。

这里通过示例来说明，在 app.json 文件 pages 页面数组中，添加 circle 页面路径，开发者工具会自动创建一个 circle 页面（circle.js、circle.json、circle.wxml、circle.wxss），这样就会显示 circle 页面。代码如图 2-6 所示。

circle.wxml 文件代码如图 2-7 所示。

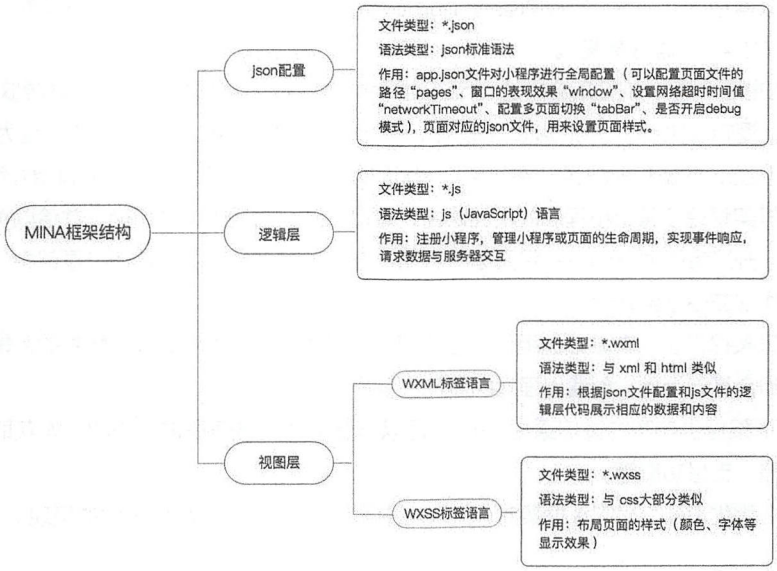


图 2-5 小程序架构（MINA 架构）

app.json

```
{
  "pages": [
    "pages/circle/circle",
    "pages/logs/logs",
    "pages/index/index"
  ],
  "window": {
    "backgroundTextStyle": "light",
    "navigationBarBackgroundColor": "#fff",
    "navigationBarTitleText": "WeChat",
    "navigationBarTextStyle": "black"
  }
}
```

图 2-6 app.json 文件代码

circle.wxml

```
<!--circle.wxml-->
<view> {{name}}! </view>

<button bindtap="changeName"> 单击 </button>
```

图 2-7 circle.wxml 文件代码

circle.js 文件代码如图 2-8 所示。

circle.json 文件代码如图 2-9 所示。

circle.js

```
//circle.js
var nameData = {
  name: '2.4 小程序架构'
}
Page({
  data: nameData,
  changeName: function(e) {
    // sent data change to view
    this.setData({
      name: '2.4.1 框架介绍'
    })
  }
})
```

图 2-8 circle.js 文件代码

index.json

```
{}
```

图 2-9 circle.json 文件代码

通过这个简单的例子我们可以看到：

逻辑层数据中的 name 与视图层的 name 通过框架进行了绑定，所以在页面一打开的时候会显示“2.4 小程序架构”。

当单击按钮的时候，视图层会发送 changeName 的事件给逻辑层，逻辑层找到对应的事件处理函数，逻辑层执行了 setData 的操作，将 name 从“2.4 小程序架构”变为“2.4.1 框架介绍”，因为该数据和视图层已经绑定了，从而视图层会自动改变为“2.4.1 框架介绍”。

MINA 框架管理了整个小程序的页面路由，可以做到页面间的无缝切换，并给以页面完整的生命周期。开发者需要做的只是将页面的数据、方法、生命周期函数注册进框架中，其他的一切复杂操作都交由框架处理。

MINA 框架提供了一套基础的组件，这些组件自带微信风格的样式以及特殊的逻辑，开发者可以通过组合基础组件，创建出强大的微信小程序。

MINA 框架提供丰富的微信原生 API，可以方便地调用微信提供的功能，如获取用户信息、本地存储、支付功能等。

基于以上优化设计，MINA 框架对 Android 端和 iOS 端做出了高度一致的呈现。

2.4.2 逻辑层

逻辑层将数据进行处理后发送给视图层，同时接收视图层的事件反馈。对微信小程序来说，逻辑层就是所有的 js 文件的集合。js 文件是由 JavaScript 编写，在 JavaScript 的基础上，微信增加和修改了以下特性：

- 增加 App 和 Page 方法，进行程序和页面的注册。
- 增加 getApp 和 getCurrentPage 方法，用来获取 App 实例和当前页面。
- 提供丰富的 API，如微信用户数据、扫一扫、支付等微信特有功能。
- 每个页面有独立的作用域，并提供模块化能力。

由于框架并非运行在浏览器中，所以 JavaScript 在 web 中的一些功能无法使用，如 document、window 等。实现逻辑层就是编写所有 js 文件，所有代码最终将会打包成一份 JavaScript，并在小程序启动的时候运行，直到小程序销毁。其功能类似于 ServiceWorker，所以逻辑层也称为 App Service。

1）小程序的生命周期（app.js）

所谓生命周期，就是指应用从启动到关闭这一过程中所发生的一系列事件。在原生的 iOS 和 Android 开发中都有类似的概念。比如 iOS 下 Application 管理 App 的生命周期，每个 view 界面都拥有生命周期。同样，在 Android 中 Application 管理安卓程序的生命周期，在 Android 页面 Activity 和 Fragment 都是拥有生命周期的。

微信小程序也有生命周期，小程序通过 app.js 来管理小程序的生命周期，同样小程序的页面 Page 也拥有自己的生命周期。

App() 函数用来注册一个小程序。该函数接收一个 object 参数，其指定小程序的生命周期函数等。App() 必须在 app.js 中注册，且不能注册多个。所以 App()函数在一个小程序中仅有一个。

object 参数说明如表 2-6 所示。

表 2-6　　object 参数说明

属性	类型	描述	触发时机
onLaunch	Function	生命周期函数——监听小程序初始化	当小程序初始化完成时，会触发 onLaunch（全局只触发一次）
onShow	Function	生命周期函数——监听小程序显示	当小程序初始化完成后会触发 onShow，或从后台进入前台显示也会触发 onShow
onHide	Function	生命周期函数——监听小程序隐藏	当小程序从前台进入后台，会触发 onHide（按后退按钮，或按 Home 键切换回桌面，这时 onHide 方法都会被调用）
其他	Any	开发者可以添加任意的函数或数据到 object 参数中	用 this 可以访问

单击左上角关闭按钮，或者按设备 Home 键离开微信，小程序并没有直接销毁，而是进入了后台；当再次进入微信或再次打开小程序时，小程序会从后台进入前台。只有当小程序进入后台一段时间后，或者系统资源占用过高时，才会被真正地销毁。

以上就是小程序目前开放的所有生命周期方法了。开发者可以根据小程序业务逻辑，来使用这些生命周期方法。

app.js 文件生命周期函数，完整的代码如图 2-10 所示。

将原有的 app.js 中的代码替换为如图 2-10 所示的代码，运行小程序，就可以在 Log 信息中看到以下 Log 信息（日志），如图 2-11 所示。

app.js

```
//app.js
App({
  onLaunch: function () {
    //小程序启动执行的初始化方法，首次打开小程序执行，只执行一次
    console.log('App onLaunch');
    //调用API从本地缓存中获取数据
    var logs = wx.getStorageSync('logs') || []
    logs.unshift(Date.now())
    wx.setStorageSync('logs', logs)
  },
  onShow: function() {
    // 小程序启动或从后台进入到前台时，执行该操作
    console.log('App onShow');
  },
  onHide: function() {
    // 小程序从前台进入到后台时，执行该操作
    console.log('App onHide');
  },
  getUserInfo:function(cb){
    var that = this
    if(this.globalData.userInfo){
      typeof cb == "function" && cb(this.globalData.userInfo)
    }else{
      //调用登录接口
      wx.login({
        success: function () {
          wx.getUserInfo({
            success: function (res) {
              that.globalData.userInfo = res.userInfo
              typeof cb == "function" && cb(that.globalData.userInfo)
            }
          })
        }
      })
    }
  },
  globalData:{
    userInfo:null
  }
})
```

图 2-10　app.js 文件代码

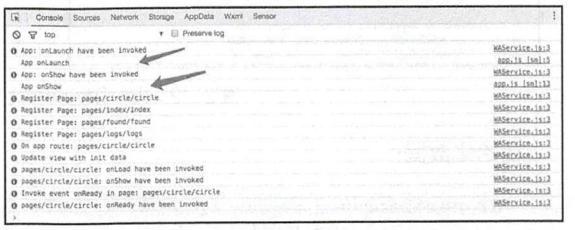


图 2-11　运行日志

微信给我们提供了 getApp() 函数，这是一个全局函数，在任何地方调用这个函数都可以得到小程序的实例来使用。这样在需要使用小程序对象相关属性的时候，就可以引用到它了。另外，利用 getCurrentPage()函数用户可以获取当前页面的实例。例如：

```
// other.js
var appInstance = getApp()
console.log(appInstance.globalData) // 输出小程序实例的 globalData 属性值
```

在使用的时候，需要注意几点：

（1）App() 必须在 app.js 中注册，且不能注册多个；

（2）不要在定义于 App() 内的函数中调用 getApp()，使用 this 就可以拿到 app 实例；

（3）不要在 onLaunch 的时候调用 getCurrentPage()，此时 page 还没有生成；

（4）getApp() 获取实例之后，不要自己调用生命周期函数。

2）页面的生命周期

在页面的 js 文件中，会看到小程序中另外一个重要的对象 Page，Page() 函数用来注册一个页面。该函数接收一个 object 参数，用来实现页面的初始数据、生命周期函数、事件处理函数等。具体代码如图 2-12 所示。

circle.js

```
//circle.js
var nameData = {
  name: '2.4小程序架构'
}
Page({
  data: nameData,
  changeName: function(e) {
    // sent data change to view
    this.setData({
      name: '2.4.1 框架介绍'
    })
  },
  changePage: function() {
    wx.navigateTo({
      url: '../logs/logs'
    })
  },
  /**
   * 生命周期函数
   */
  onLoad: function () {
    // 生命周期函数----页面加载时触发
    console.log('circle---------onLoad');
  },
  onShow: function() {
    // 生命周期函数----页面显示时触发
    console.log('circle---------onShow');
  },
  onReady: function() {
    // 生命周期函数----页面初次渲染完成时触发，完成之后不在执行
    console.log('circle---------onReaday');
  },
  onHide: function() {
    // 生命周期函数----页面隐藏（页面跳转到新页面）时触发
    console.log('circle---------onHide');
  },
  onUnload: function() {
    // 生命周期函数----页面关闭（卸载、销毁）时触发
    console.log('circle---------onUnload');
  },
  onPullDownRefresh: function() {
    // 页面事件函数----下拉事件
    console.log("page onPullDownRefresh");
  onHide: function() {
    // 生命周期函数----页面隐藏（页面跳转到新页面）时触发
    console.log('index---------onHide()')
  },
  onUnload: function() {
    // 生命周期函数----页面关闭（卸载、销毁）时触发
    console.log('index---------onUnload')
  },
  onPullDownRefresh: function() {
    // 页面事件函数----下拉事件
    console.log("page onPullDownRefresh");
  },
  onReachBottom: function() {
    // 页面事件函数----上拉事件
    console.log("page onReachBottom");
  },
  onShareAppMessage: function() {
    // 用户单击右上角分享
    return {
      title: '微信小程序开发', // 分享标题
      desc: '快速掌握小程序项目的开发，实现一本书搞定小程序开发', // 分享描述
      path: 'pages/circle/circle' // 分享路径
    }
  }
})
```

图 2-12 Page()的 object 参数代码

初始化数据：

```
var nameData = {
  name: '2.4 小程序架构'
}
```

```
data: nameData,
```

初始化数据将作为页面的第一次渲染显示内容。data 将会以 JSON 的形式由逻辑层传至渲染层，所以其数据必须是可以转成 JSON 的格式：字符串、数字、布尔值、对象、数组。渲染层可以通过 WXML 对数据进行绑定。

WXML 代码如下所示：

```
<!--circle.wxml-->
<view> {{name}}! </view>
```

生命周期函数：

onLoad：页面加载，一个页面只会调用一次。

参数可以获取 wx.navigateTo 和 wx.redirectTo 及<navigator/>中的 query。

onShow：页面显示，每次打开页面都会调用一次。

onReady：页面初次渲染完成，一个页面只会调用一次，代表页面已经准备妥当，可以和视图层进行交互。

对页面的设置（如 wx.setNavigationBarTitle）请在 onReady 之后设置。

onHide：页面隐藏，当 navigateTo 或底部 tab 切换时调用。

onUnload：页面卸载，当 redirectTo 或 navigateBack 的时候调用。

将 circle 页面设置为首页（在 app.json 中设置），程序会自动加载 circle 页面，调用 circle.js 中的生命周期方法，打印 Log 信息（日志），如图 2-13 所示。

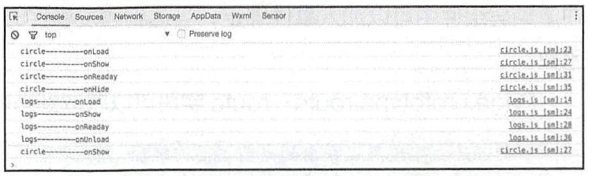


图 2-13 circle.js 中的生命周期输出日志

页面相关事件处理函数：

onPullDownRefresh：下拉刷新事件。开发者需要在 config 的 window 选项中开启 enablePullDownRefresh 才可使用该函数。当处理完数据刷新后，wx.stopPullDownRefresh 可以停止当前页面的下拉刷新。使用代码如下所示：

```
Page({

  onPullDownRefresh: function () {
    console.log('onPullDownRefresh' + new Date())
  },
  stopPullDownRefresh: function () {
    wx.stopPullDownRefresh({
      complete: function (res) {
        console.log(res + new Date())
      }
    })
  }
})
```

onReachBottom：上拉刷新事件。页面上拉触发底事件的处理函数。

onShareAppMessage：用户单击右上角转发事件。只有定义了此事件处理函数，右上角菜单才会显示“转发”按钮，用户单击转发按钮的时候会调用，此事件需要 return 一个 Object，用于自定义转发内容。用户可以自定义 title（转发标题）、path（转发路径，当前页面 path，必须是以 / 开头的完整路径）。

示例代码如下所示：

```
Page({
onShareAppMessage: function ()
{
  return
  {
     title: '自定义转发标题',
     path: '/page/user?id=123'
   }
}
})
```

onPageScroll：页面滚动触发事件的处理函数，参数是 scrollTo（Number 类型），表示页面在垂直方向已滚动的距离（单位为 px）。

事件处理函数：

除了初始化数据和生命周期函数，Page 中还可以定义一些特殊的函数：事件处理函数。在渲染层组件中可以加入事件绑定，当达到触发事件时，就会执行 Page 中定义的事件处理函数。

Page.prototype.route：route 字段可以获取到当前页面的路径。代码如下所示：

```
//route 字段可以获取到当前页面的路径。
   console.log('当前路径是: ' + this.route);
```

Page.prototype.setData()：setData()函数用于将数据从逻辑层发送到视图层，同时改变对应的 this.data 的值。

setData() 参数接收一个对象，以 key-value 的形式表示将 this.data 中的 key 对应的值改变成 value。

注意：

（1）直接修改 this.data 而不调用 this.setData 是无法改变页面的状态的，还会造成数据不一致；

（2）单次设置的数据大小不能超过 1024KB，请尽量避免一次设置过多的数据。

代码如下所示：

```
//circle.js
changeName: function(e) {
   // sent data change to view
   this.setData({
```

```
      name: '2.4.1 框架介绍'
    })
  },
  changePage: function() {
    wx.navigateTo({
      url: '../logs/logs'
    })
  },

<!--circle.wxml-->
<view> {{name}}! </view>

<button bindtap="changeName"> 单击 </button>
<button bindtap="changePage"> 查看日志 </button>
```

3）文件作用域和模块化

js（JavaScript）页面文件中声明的变量和函数值在该文件中有效，不同的文件中可以声明相同名字的变量和函数，不会互相影响。

如果需要全局的数据可以在 App()中设置，代码如图 2-14 所示。

在 index.js 文件中定义局部变量 localCount 等于 1，代码如图 2-15 所示。

app.js

```
1  //app.js
2  App({
3    //定义全局变量globalCount
4    globalCount:1,
5    globalData:{
6      userInfo:null
7    },
8    onLaunch: function () {
9      //调用API从本地缓存中获取数据
10     var logs = wx.getStorageSync('logs') || []
11     logs.unshift(Date.now())
12     wx.setStorageSync('logs', logs)
13   },
```

图 2-14 app.js 定义全局变量

index.js

```
1  //index.js
2  //获取应用实例
3  var app = getApp()
4
5  //定义局部变量
6  var localCount = 1
7
8  Page({
9    data: {
10     motto: 'Hello World',
11     userInfo: {}
12   },
```

图 2-15 index.js 定义局部变量

在 circle.js 文件中定义局部变量 localCount 等于 2，并修改全局变量，代码如图 2-16 所示。

通过全局函数 var app = getApp()，开发者可以获取全局的应用实例，每次触发 changeCount 函数时，全局变量 app.globalCount 都会加 1。虽然 index.js 文件中 localCount 等于 1，但不影响本文件定义 localCount 等于 2。开发者可以从 Console 页看到全局变量和局部变量的输出结果，如图 2-17 所示。

每个页面都会有对应的 js，我们可以只单独创建一个 js 文件作为一个模块，类似于 iOS 开发中的一个类。提供函数给其他页面使用，小程序 js 模块只有通过 module.exports 暴露接口，其他 js 页面才可以引入引用。

在新创建的项目中，都会有 util.js 文件，文件定义了“formatTime”方法和“formatNumber”方法，但是 module.exports 中只暴露了“formatTime”方法供其他 js 文件使用。文件代码如图 2-18 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

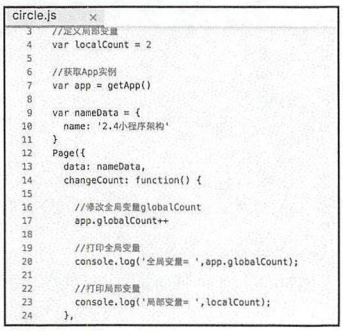


图 2-16 circle.js 定义局部变量和修改全局变量

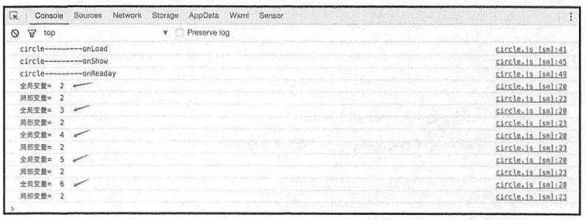


图 2-17 changeCount 函数执行日志输出信息

在需要使用 util.js 模块函数的 js 文件中，使用 require（path）引入模块（初始化模块的过程），然后才可以使用模块的方法。代码如图 2-19 所示。

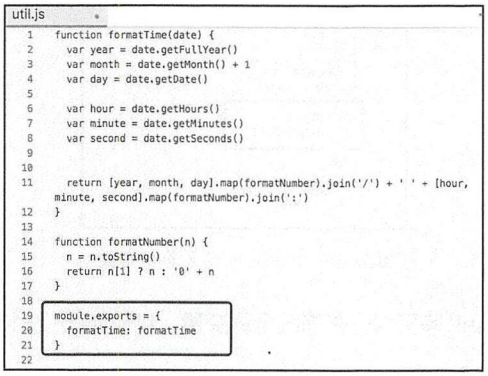


图 2-18 util.js 文件代码

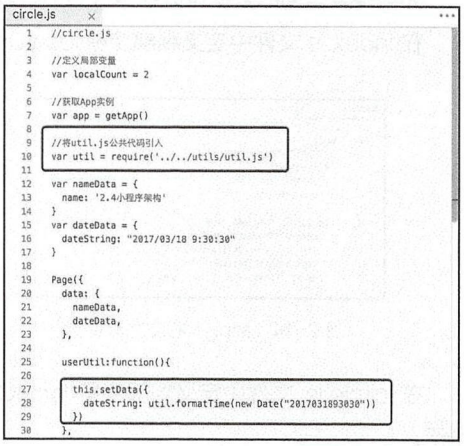


图 2-19 使用 util.js 模块

2.4.3 视图层（WXML 和 WXSS 介绍）

- MINA 的视图层由 WXML（WeiXin Markup Language）与 WXSS（WeiXin Style Sheet）编写，由组件显示内容。MINA 将逻辑层的数据反应成视图，同时将视图层的事件发送给逻辑层。
- WXML 是用于描述页面的结构、框架设计的一套标签语言。
- WXSS 用于描述页面的样式。
- 组件（Component）是视图的基本组成单元。

也就是说 WXML 通过各种组件来展现内容，WXSS 提供页面 CSS 样式。WXML 所使用

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

绑定的数据，来自页面对应的 js 文件中 Page 方法中的 data 对象。

关于 WXML 和 WXSS，我们将在 2.5 节、2.6 节进行详细介绍。

2.4.4 数据层

关于数据层的一些基本应用，在 2.4.2 节其实已经提过。一个是 App({...})里定义的 globalCount，用来全局共享数据；另一个是每个 Page 自己的数据字段 data：{ ... }。

数据层主要包括：本地数据操作、本地存储、文件操作、网络数据请求和调用。

1）本地数据操作

本地数据操作主要是指页面临时数据或缓存，在 Page()中，要使用 setData()函数将数据从逻辑层发送到视图层进行显示，同时改变对应的 this.data 中的值。不能直接修改 this.data，修改之后视图层无法刷新显示。另外，微信规定单次设置的数据大小不能超过 1024KB。

setData()函数的参数接收一个对象。对象以 key-value 形式表示，setData()函数将 this.data 中的 key 对应的值改变成 value。其中 key 可以非常灵活，可以是字符串，可以是对象，还可以是数组的属性，例如 array[0].text，等等。

创建一个 demoSetData 页面（同时创建 demoSetData.js、demoSetData.json、demoSetData. wxml、demoSetData.wxss）。

demoSetData.wxml 文件代码如图 2-20 所示。

demoSetData.wxml 页面定义了 4 个 view，分别绑定了普通文本（text）、数组文本（array[0].text）、对象文本（object.text）、动态新对象文本（newField.text）变量。另外再定义 4 个按钮，绑定对应的时间，分别修改上面 4 种变量。

demoSetData.js 文件代码如图 2-21 所示。

demoSetData.js 文件中，data 数组初始化了 3 种不同类型的数据，下面事件中分别使用 this.setData()函数来重新设置对应类型的数据。addNewField 事件方法通过一个新的 key “newField.text”，给 data 数据增加了一种新对象类型的数据，并给它赋值“新对象”。大家可以从下载 demo 源码中，添加 CH2.4 项目，运行查看效果。

通过上面的例子，可以看到给数组 array 修改值的时候，我们是修改数组中下标 0 的文本内容，在实际开发中，我们对数组中的某个元素的设置是动态的。即通常应该是 ‘array[‘+index+’].text’ : ‘changed data’，其中 index 是一个动态的数字。这样直接写 index，是无法使用在对象中的 key 的。我们可以使用一个变通的方法，先计算出 index 值，再对数组下表进行操作，代码如下所示：

```
var index = m+n;
this.data.array[index]. text = "new text";
this.setData(
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    {
      array: this.data. array
    });
```

```
demoSet...
1  <!--demoSetData.wxml-->
2  <view>{{text}}</view>
3  <button bindtap="changeText"> 修改字符串 </button>
4  <view>{{array[0].text}}</view>
5  <button bindtap="changeItemInArray"> 修改数组 </button>
6  <view>{{object.text}}</view>
7  <button bindtap="changeItemInObject"> 修改对象 </button>
8  <view>{{newField.text}}</view>
9  <button bindtap="addNewField"> 添加一个新对象 </button>
```

图 2-20 demoSetData.wxml 文件代码

```
demoSet...
1  //demoSetData.js
2  Page({
3    data: {
4      text: '字符串',
5      array: [{text: '数组内字符串'}],
6      object: {
7        text: '对象内字符串'
8      }
9    },
10   changeText: function() {
11     // this.data.text = 'changed data'  // bad, it can not work
12     this.setData({
13       text: '字符串--修改成功'
14     })
15   },
16   changeItemInArray: function() {
17     // you can use this way to modify a danamic data path
18     this.setData({
19       'array[0].text':'数组内字符串--修改成功'
20     })
21   },
22   changeItemInObject: function(){
23     this.setData({
24       'object.text': '对象内字符串--修改成功'
25     });
26   },
27   addNewField: function() {
28     this.setData({
29       'newField.text': '新对象'
30     })
31   }
32 })
```

图 2-21 demoSetData.js 文件代码

在介绍本地存储、文件操作和网络数据请求之前，我们先来了解一下微信原生 API。框架提供丰富的微信原生 API，可以方便地调用微信提供的功能，如获取用户信息、本地存储、支付功能等。

微信 API 使用说明：

wx.* 开头的 API 是监听某个事件发生的 API 接口，接收一个 callback()函数作为参数。当该事件触发时，会调用 callback()函数。如未特殊约定，其他 API 接口都接收一个 OBJECT 作为参数。OBJECT 中可以指定 success、fail、complete 来接收接口调用结果，如表 2-7 所示。

表 2-7 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

API 分为：网络 API、媒体 API、数据 API、位置 API、设备 API、界面 API、开放接口，如表 2-8 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 2-8 API

网络 API	
API	说明
wx.request	发起网络请求
wx.uploadFile	上传文件
wx.downloadFile	下载文件
wx.connectSocket	创建 WebSocket 连接
wx.onSocketOpen	监听 WebSocket 打开
wx.onSocketError	监听 WebSocket 错误
wx.sendSocketMessage	发送 WebSocket 消息
wx.onSocketMessage	接收 WebSocket 消息
wx.closeSocket	关闭 WebSocket 连接
wx.onSocketClose	监听 WebSocket 关闭
媒体 API	
API	说明
wx.chooseImage	从相册选择图片，或者拍照
wx.previewImage	预览图片
wx.startRecord	开始录音
wx.stopRecord	结束录音
wx.playVoice	播放语音
wx.pauseVoice	暂停播放语音
wx.stopVoice	结束播放语音
wx.getBackgroundAudioPlayerState	获取音乐播放状态
wx.playBackgroundAudio	播放音乐
wx.pauseBackgroundAudio	暂停播放音乐
wx.seekBackgroundAudio	控制音乐播放进度
wx.stopBackgroundAudio	停止播放音乐
wx.onBackgroundAudioPlay	监听音乐开始播放
wx.onBackgroundAudioPause	监听音乐暂停
wx.onBackgroundAudioStop	监听音乐结束
wx.chooseVideo	从相册选择视频，或者拍摄
wx.saveFile	保存文件
数据 API	
API	说明
wx.getStorage	获取本地数据缓存
wx.setStorage	设置本地数据缓存
wx.clearStorage	清理本地数据缓存

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

续表

位置 API	
API	说明
wx.getLocation	获取当前位置
wx.openLocation	打开内置地图
设备 API	
API	说明
wx.getNetworkType	获取网络类型
wx.getSystemInfo	获取系统信息
wx.onAccelerometerChange	监听重力感应数据
wx.onCompassChange	监听罗盘数据
界面 API	
API	说明
wx.setNavigationBarTitle	设置当前页面标题
wx.showNavigationBarLoading	显示导航条加载动画
wx.hideNavigationBarLoading	隐藏导航条加载动画
wx.navigateTo	新窗口打开页面
wx.redirectTo	原窗口打开页面
wx.navigateBack	退回上一个页面
wx.createAnimation	动画
wx.createContext	创建绘图上下文
wx.drawCanvas	绘图
wx.hideKeyboard	隐藏键盘
wx.stopPullDownRefresh	停止下拉刷新动画
开放接口	
API	说明
wx.login	登录
wx.getUserInfo	获取用户信息
wx.requestPayment	发起微信支付

各接口详情介绍，请参考第四章、第五章内容。

2）本地存储

微信给每个小程序分配了 10MB 的存储空间，对这个缓存的操作，其实跟 H5 的 localStorage 操作是一样的。微信提供了 3 个主要的接口：wx.setStorage（wx.setStorageSync）、wx.getStorage（wx.getStorageSync）、wx.clearStorage（wx.clearStorageSync），括号里面的方法是同步读写的方法，括号外面的方法是异步读写的方法。这两类方法调用时传入的参数大体差

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

不多，只是异步方法是通过传入回调函数的方式来取到数据，而同步方法是直接返回数据。

- 存储的数据比较大的时候，同步方法会引起界面的卡顿，所以用异步方法好些，不影响界面。
- 异步方法的错误处理是 fail 回调，出错一般是通过 fail 传出来的。而同步方法要加上 try catch 才能避免程序异常中断。
- 同步方法比异步方法在使用的时候简单很多。如果存储的数据很小，基本确定不会有什么逻辑错误，可以直接用同步方法。但如果觉得你的应用有可能会达到 5MB 的空间时，再存就会报错，这时使用同步方法就要注意 try catch。

（1）存储

异步存储方法 wx.setStorage，参数是 json，字段用 key 指定存储的键值，data 指定存储的值，代码如下所示：

```
//异步存储简单写法
wx.setStorage({ key: 'name', data: 'LiuMingyang' });
```

完整的异步存储格式是要传入 success、fail 和 complete 来取得数据或错误信息，代码如下所示：

```
//异步存储
   wx.setStorage({
     key: 'name',
     data: 'LiuMingyang',
     success: function (res) {
       // success
     },
     fail: function () {
       // fail
     },
     complete: function () {
       // complete
     }
   });
```

而同步存储方法 wx.setStorageSync 比较简单，参数是直接传的，代码如下所示：

```
//同步存储
try {
     wx.setStorageSync('sex', 'male')
   } catch (e) {
   }
```

存入的 data 数据可以是字符串，也可以是 json 格式的对象 Object。如果存的是 json 格式的，取的时候也会是 json 格式的。这个用起来会非常方便。存入的数据，可以在调试工具的 Storage 里看到，如图 2-22 所示。

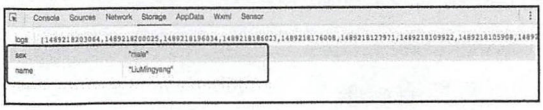


图 2-22 Storage 信息

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

（2）取出

异步取出方法 wx.getStorage 也是传入 json，字段只要带一个 key 字符串就行，通过字段 success 回调来取得参数，代码如下所示：

```
//异步取出
wx.getStorage({
  key: 'name',
  success: function (res) {
    // success
    console.log("name = ", res.data)
  },
  fail: function () {
    // fail
  },
  complete: function () {
    // complete
  }
})
```

而同步取出方法 wx.getStorageSync 就非常简单，直接就可以取到返回值，代码如下所示：

```
//同步取出
try {
  var value = wx.getStorageSync('sex')
  console.log("sex = ", value)
} catch (e) {
}
```

（3）删除

直接调用 wx.clearStorage() 或 wx.clearStorageSync()就成。不过调用这个方法时要注意，它会清除掉所有的数据。

（4）删除单个 key

要删除单个 key 的数据也很简单，wx.removeStorage 和 wx.removeStorageSync 使用方式跟 wx.getStorage 和 wx.getStorageSync 一样，代码如下所示：

```
//异步删除
wx.removeStorage({
  key: 'name',
  success: function (res) {
    // success
  },
  fail: function () {
    // fail
  },
  complete: function () {
    // complete
  }
})

//同步删除
try {
  wx.removeStorageSync('sex')
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    } catch (e) {
    }
```

（5）查看缓存空间的所有信息，代码如下所示：

```
//查看缓存空间的所有信息
    //异步查看
    wx.getStorageInfo({
      success: function (res) {
        // success
        console.log(res.keys)
        console.log(res.currentSize)
        console.log(res.limitSize)
      },
      fail: function () {
        // fail
      },
      complete: function () {
        // complete
      }
    })

    //同步查看
    try {
      var res = wx.getStorageInfoSync()
      console.log(res.keys)
      console.log(res.currentSize)
      console.log(res.limitSize)
    } catch (e) {
      //做同样的事
    }
```

返回信息包括 3 个数据项，分别是：

keys：当前 Storage 中存储的所有内容的 key 集合。

currentSize：当前缓存所占空间的大小，单位是 KB。

limitSize：限制的最大空间，单位是 KB（1024KB）。

3）文件操作

（1）下载文件

读取文件或者删除文件等操作的前提是本地有文件，所以首先下载一个文件，代码如下所示：

```
//下载文件
    wx.downloadFile({
      url: "http://91ruankao.com/wx.doc",
      // type: 'image', // 下载资源的类型，用于客户端识别处理，有效值：image/
      //audio/video
      // header: {}, // 设置请求的 header
      success: function (res) {
        // success

        console.log("downloadFile------success")

        //获取下载文件路径
        var filePath = res.tempFilePath
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        console.log("filePath------", filePath)

        //保存文件到本地
        lmy_SaveFile(filePath)

        //打开文件
        lmy_openDocument(filePath)
      },
      fail: function () {
        // fail
        console.log("downloadFile------fail")
      },
      complete: function () {
        // complete
        console.log("downloadFile------complete")
      }
    })
  },
```

上面的代码成功下载文件后，调用了 lmy_SaveFile 方法保存到本地，lmy_openDocument 方法打开文件。

（2）保存文件

微信提供的保存文件到本地的 API 是 wx.saveFile，具体代码如下所示：

```
//保存文件到本地
function lmy_SaveFile(filePath) {
  wx.saveFile({
    tempFilePath: filePath,
    success: function (res) {
      // success
      console.log("saveFile------success", res)
    },
    fail: function () {
      // fail
      console.log("saveFile------fail")
    },
    complete: function () {
      // complete
      console.log("saveFile------complete")
    }
  })
}
```

（3）打开文件

微信提供的打开本地文件的 API 是 wx.openDocument，具体代码如下所示：

```
//打开文件
function lmy_openDocument(filePath) {
  wx.openDocument({
    filePath: filePath,
    success: function (res) {
      // success
      console.log("openDocument------success", res)
    },
    fail: function () {
```

```
      // fail
      console.log("openDocument------fail", res)
    },
    complete: function () {
      // complete
      console.log("openDocument------complete", res)
    }
  })
}
```

（4）获取保存的所有文件列表

wx.getSavedFileList 用来获取本地已保存的文件列表，代码如下所示：

```
wx.getSavedFileList({
    success: function (res) {
      // success
      console.log("getSavedFileList------success", res)
    },
    fail: function () {
      // fail
      console.log("getSavedFileList------fail")
    },
    complete: function () {
      // complete
      console.log("getSavedFileList------complete")
    }
  })
```

从打印出来的信息可以看到 res 对象有个 fileList 数组，数组里面存储了文件路径，路径格式为：

```
wxfile://xxxxxxxxxxxxxxx.doc
```

（5）获取文件信息

wx.getSavedFileInfo 用来获取文件的信息（文件的大小、创建日期等），代码如下所示：

```
//读取文件信息
  wx.getSavedFileInfo({
    filePath: savedFilePath,
    success: function (res) {
      // success
      console.log("getSavedFileInfo------success", res)
    },
    fail: function () {
      // fail
      console.log("getSavedFileInfo------fail")
    },
    complete: function () {
      // complete
      console.log("getSavedFileInfo------complete")
    }
  })
```

（6）删除文件

wx.removeSavedFile 用来删除本地已保存的文件，传入文件路径即可删除，代码如下所示：

```
//删除本地已保存的文件
  wx.removeSavedFile({
    filePath: savedFilePath,
    success: function (res) {
      // success
      console.log("removeSavedFile------success")
    },
    fail: function () {
      // fail
      console.log("removeSavedFile------fail")
    },
    complete: function () {
      // complete
      console.log("removeSavedFile------complete")
    }
  })
```

4）网络数据请求

微信提供了一个接口用来发起请求：wx.request。而且它发起的请求，只能是 HTTPS 请求，且一个微信小程序，同时只能有 5 个网络请求连接。进行网络通信时，小程序只能和指定的域名进行通信。微信小程序包括 4 种类型的网络请求：

普通 HTTPS 请求（wx.request）

上传文件（wx.uploadFile）

下载文件（wx.downloadFile）

WebSocket 通信（wx.connectSocket）

上传或下载文件 API 接口，如表 2-9 所示。

表 2-9　　上传或下载文件 API 接口

API	说明
wx.request	发起网络请求
wx.uploadFile	上传文件
wx.downloadFile	下载文件

5）调用

微信小程序通过 navigateTo 或者 redirectTo 实现连接跳转时的参数传值。

调用 URL 的 API 接口，如表 2-10 所示。

表 2-10　　调用 URL 的 API 接口

API	说明
wx.navigateTo	新窗口打开页面
wx.redirectTo	原窗口打开页面

2.5　视图层 WXML 介绍

WXML（WeiXin Markup Language）是微信的一套标签语言（类似 HTML），结合基

础组件、事件系统，可以构建出页面的结构。WXML 在项目中以.wxml 文件呈现。index.wxml 文件代码如图 2-23 所示。

index.wxml

```
<!--index.wxml-->
<view class="container">
  <view  bindtap="bindViewTap" class="userinfo">
    <image class="userinfo-avatar" src="{{userInfo.avatarUrl}}"
background-size="cover"></image>
    <text class="userinfo-nickname">{{userInfo.nickName}}</text>
  </view>
  <view class="usermotto">
    <text class="user-motto">{{motto}}</text>
  </view>
</view>
```

图 2-23 index.wxml 代码

文件第一行，写<!--index.wxml-->注释，然后是页面的呈现部分。<view>组件使页面显示出内容，<text>组件实现页面数据的绑定。

WXML 具有数据绑定、列表渲染、条件渲染、模板、事件绑定的功能。下面将分别介绍这些功能。开发者可以参考 Ch2.5 项目工程。

1）数据绑定

WXML 中的动态数据均来自对应页面 js 文件中 Page 的 data。数据绑定使用 Mustache 语法（双大括号）将变量包起来，用于内容、组件属性、控制属性、关键字等场景。

（1）数据的简单绑定

数据作为内容使用，代码如下所示：

```
<!--dataBinding.wxml-->
<view> {{text}} </view>

// dataBinding.js
Page({
    data: {
        text: '欢迎学习微信小程序开发!'
    }
})
```

数据作为组件属性（需要在双引号之内）使用，比如动态定义控件 view 的 id，代码如下所示：

```
<!--dataBinding.wxml-->
<view id="item-{{id}}"> </view>

// dataBinding.js
Page({
  data: {
    id: 0
  }
})
```

数据作为控制属性（需要在双引号之内）使用，条件判断是否显示控件，代码如下所示：

```
<!--dataBinding.wxml-->
<view wx:if="{{condition}}"> 欢迎学习微信小程序开发 </view>

// dataBinding.js
Page({
  data: {
    condition: false
```

```
  }
})
```

数据作为关键字（需要在双引号之内）使用，代码如下所示：

```
<!--dataBinding.wxml-->
<checkbox checked="{{true}}"> </checkbox>-->
```

不要直接写 checked="false"，其计算结果是一个字符串，需要转成 Boolean 类型后才能代表真值。

true：Boolean 类型的 "{{true}}"代表真值。false：Boolean 类型的 "{{false}}"代表假值。

（2）数据的运算

数据绑定的时候，可以在{{ }}内进行简单的运算，支持三元运算、算数运算、逻辑判断、字符串运算、数据路径运算几种方式。

三元运算，代码如下所示：

```
<!--dataBinding.wxml-->
<view hidden="{{(1>0) ? true : false}}"> 欢迎学习微信小程序开发 </view>
```

算数运算，代码如下所示：

```
<!--dataBinding.wxml-->
<!--算数运算-->
<view> {{a + b}} + {{c}} + d </view>

// dataBinding.js
Page({
  data: {
    a: 1,
    b: 2,
    c: 3
  }
})
```

运行上面的代码，界面显示：3 + 3 + d。

逻辑判断，判断 count>5 时，才显示控件，代码如下所示：

```
<!--dataBinding.wxml-->
<!--逻辑判断-->
<view wx:if="{{count > 5}}"> 欢迎学习微信小程序开发 </view>

// dataBinding.js
Page({
  data: {
    count: 6
  }
})
```

字符串运算，用于拼接字符串，代码如下所示：

```
<!--dataBinding.wxml-->
```

```
<!--字符串运算-->
<view>{{"hello " + name}}</view>

// dataBinding.js
//字符串运算
Page({
  data:{
    name:"MINA"
  }
})
```

数据路径运算，显示对象属性或数组中的值，代码如下所示：

```
<!--dataBinding.wxml-->
<!--数据路径运算-->
<view>{{object.key}} {{array[0]}}</view>

// dataBinding.js
Page({
  data: {
    object: {
      key: 'Hello '
    },
    array: ['MINA']
  }
})
```

（3）数据的组合

数据可以在 Mustache 内直接进行变量组合，构成新的对象或者数组。

数组的组合，代码如下所示：

```
<!--dataBinding.wxml-->
<!--数据的组合-->
<view wx:for-items="{{[value, 1, 2, 3, 4]}}"> {{item}} </view>

// dataBinding.js
Page({
  data: {
    value: 0
  }
})
```

数组中第一个是变量 value，通过 data 赋值来组成数组，最终组合成的数组是[0, 1, 2, 3, 4]。

对象组合，代码如下所示：

```
<!--dataBinding.wxml-->
<!--对象组合-->
<template name="objectCombine">
  <view>
    <text> for = {{for}}</text>
    <text> bar = {{bar}} </text>
  </view>
</template>
<template is="objectCombine" data="{{for: x, bar: y}}"></template>
// dataBinding.js
```

```
//对象组合
Page({
  data: {
    x: 1,
    y: 2
  }
})
```

上面的代码通过 data 赋值，组合成的模板对象是：{for: 1, bar: 2}。利用 objectCombine 模板，输出：for = 1 bar = 2。上面模板中 key=“for”对应的 value=“x”，key=“bar”对应的 value=“y”，如果对象的 key 和 value 相同，也可以间接地表示为：

```
<template is="objectCombine" data="{{for, bar}}"></template>
```

数据组合也用在扩展运算符...来将一个对象展开，代码如下所示：

```
<!--dataBinding.wxml-->
<!--扩展运算符-->
<template name="extendedCombine">
  <view>
    <text> a = {{a}}</text>
    <text> b = {{b}}</text>
    <text> c = {{c}}</text>
    <text> d = {{d}}</text>
    <text> e = {{e}}</text>
  </view>
</template>
<template is="extendedCombine" data="{{...obj1, ...obj2, e: 10}}"></template>

// dataBinding.js
//扩展运算符
Page({
  data: {
    obj1: {
      a: 2,
      b: 4
    },
    obj2: {
      c: 6,
      d: 8
    }
  }
})
```

extendedCombine 模板利用 data 对 obj1、obj2 赋值，扩展模板 data 数据。使用模板显示 abcd，可以看到合成的模板对象为：{a: 2, b: 4, c: 6, d: 8, e: 10}。如有存在变量名相同的情况，后面的会覆盖前面的，如：

<template is="extendedCombine" data="{{...obj1, ...obj2, c: 10}}"></template>最终组合成的模板对象是：{a: 2, b: 4, c: 10, d: 8}。

2）列表渲染

wxml 文件的列表渲染依赖于列表语句，将列表中的数据进行重复渲染。

（1）wx:for

在组件上使用 wx:for 控制属性绑定一个数组，即可使用数组中各项的数据重复渲染该组件。

如果数组中是字符串，列表渲染代码如下所示：

```
<!--listRendering.wxml-->
<view wx:for="{{array}}"> {{item}} </view>

// listRendering.js
Page({
    data: {
        array: ['第一章', '第二章', '第三章', '第四章', '第五章', '第六章', '第七章']
    }
})
```

如果数组中是“key-value”对象，列表渲染代码如下所示：

```
<!--listRendering.wxml-->
<!--key-value 对象数组-->
<view wx:for="{{items}}">
  {{index}}: {{item.fruits}}
</view>

// listRendering.js
//key-value 对象数组
Page({
      data:{
          items: [{
              fruits: 'apple',
          },{
              fruits: 'orange'
          },{
              fruits: 'banana'
          }]
      }
})
```

上面代码使用 for 循环时，数组当前项的下标变量名默认为 index，数组当前项的变量默认为 item。不使用默认变量的话，可以使用 wx:for-item 指定数组当前元素的变量名，使用 wx:for-index 指定数组当前下标的变量名，代码如下所示：

```
<!--listRendering.wxml-->
<view wx:for="{{array}}" wx:for-index="id" wx:for-item="object">
  {{id}}: {{object.fruit}}
</view>
```

wx:for 也可以嵌套，下边是一个九九乘法表，代码如下所示：

```
<!--wx:for 也可以嵌套，九九乘法表-->
<view wx:for="{{[1, 2, 3, 4, 5, 6, 7, 8, 9]}}" wx:for-item="i">
  <view wx:for="{{[1, 2, 3, 4, 5, 6, 7, 8, 9]}}" wx:for-item="j">
    <view wx:if="{{i <= j}}">
      {{i}} * {{j}} = {{i * j}}
```

```
      </view>
    </view>
  </view>
```

（2）block wx:for

因为 wx:for 是一个控制属性，所以可以添加到一个标签上控制该标签。但是如果想一次性控制多个组件标签，可以使用一个<block/>标签将多个组件包装起来，并在上边使用 wx:for 控制属性，以渲染包含多节点的结构块，代码如下所示：

```
<!--wx:for 用在<block/>标签上-->
<block wx:for="{{[1, 2, 3]}}">
  <view> {{index}}: </view>
  <view> {{item}} </view>
</block>
```

（3）唯一标识符：wx:key

如果列表中项目的位置会改变或者有新的项目添加到列表中，并且希望列表中的项目保持原有的状态（如 input 控件中的输入内容，switch 控件的选中状态），就需要使用 wx:key 来指定列表中项目的唯一标识符。

使用 wx:key 设置位置标识符之后，数据改变触发渲染层重新渲染时，框架会校正带有 key 的组件，确保它们被重新排序，而不是重新创建。这样就可以确保每个组件保持自身的状态，并且提高列表渲染时的效率。

使用 wx:key 有两种方法：

- 取 array 中 item 的某个属性作为字符串，该属性的值必须是对象唯一的字符串或数字，且不能动态改变，例如：wx:key="某个属性"。我们可以看到 wx:key 里的值并不需要双大括号，这里是比较特别的地方，不需要双大括号，同时也不需要参数名，需要的是对象数据里的一个字段名。
- 使用保留关键字 *this，它代表在 for 循环中的 item 本身，这种表示需要 item 本身是一个唯一的字符串或者数字，item 本身不会发生任何改变，例如：wx:key="*this"。

如不提供 wx:key，会报一个 warning，如果明确知道该列表是静态，或者不必关注其顺序，可以选择忽略。在开发过程中，wx:key 的作用对于项目是非常大的，如果从文字上无法理解上面内容，可以查看 Ch2.5 项目中列表渲染功能，开启 switch2 开关，在没有使用 wx:key="unique"的时候，添加一个新的 switch 后，switch1 开启，并不是 switch2 开启。如果使用 wx:key="unique"，添加一个新的 switch 后，依旧是 switch2 开启。代码如下所示：

```
    <!--listRendering.wxml-->
    <switch wx:for="{{objectArray}}" wx:key="unique" style="display: block;">
{{item.id}}</switch>
    <button bindtap="addToFront"> 在最前面添加一个 switch </button>

    <switch wx:for="{{numberArray}}" wx:key="*this" style="display: block;">
{{item}} </switch>
    <button bindtap="addNumberToFront"> 在最前面添加一个 switch </button>
```

```
// listRendering.js
Page({
  data: {
    objectArray: [
      {id: 2, unique: 'unique_2'},
      {id: 1, unique: 'unique_1'},
      {id: 0, unique: 'unique_0'},
    ],
    numberArray: [1, 2]
  },
  addToFront: function(e) {
    const length = this.data.objectArray.length
    this.data.objectArray = [{id: length, unique: 'unique_' + length}].concat
(this.data.objectArray)
    this.setData({
      objectArray: this.data.objectArray
    })
  },
  addNumberToFront: function(e){
    this.data.numberArray = [ this.data.numberArray.length + 1 ].concat(this.
data.numberArray)
    this.setData({
      numberArray: this.data.numberArray
    })
  }
})
```

3）条件渲染

.wxml 文件条件渲染主要依赖于条件语句，不同的条件语句会进行不同的渲染。

（1）wx:if

在 MINA 中，我们用 wx:if="{{条件语句}}"来判断是否需要渲染该代码块，代码如下所示：

```
<!--cRendering.wxml-->
<view wx:if="{{view == 'iOS'}}"> iOS </view>
<view wx:elif="{{view == 'Android'}}"> Android </view>
<view wx:else="{{view == 'MINA'}}"> MINA </view>

// cRendering.js
Page({
  data: {
    view: 'MINA'
  }
})
```

（2）block wx:if

因为 wx:if 是一个控制属性，所以需要将它添加到一个标签上，控制该标签。但是如果想一次性控制多个组件标签，就需要使用一个<block/>标签将多个组件包装起来，并在上边使用 wx:if 控制属性。<block/>并不是一个组件，它仅仅是一个包装元素，不会在页面中做任何渲染，只接收控制属性。

具体代码如下所示：

```
<block wx:if="{{true}}">
  <view> 标题 </view>
  <view> 描述 </view>
</block>
```

注意：

wx:if 与 hidden 功能对比：

- 由于 wx:if 之中的模板也可能包含数据绑定，所以当 wx:if 的条件值改变时，MINA 有一个局部渲染的过程，它会确保条件块在切换时被销毁或重新渲染。
- wx:if 也是惰性的，如果初始渲染条件为 false，MINA 不处理条件块，在条件第一次变成 true 的时候才会局部渲染。
- hidden 组件始终会被渲染，只是简单地控制显示与隐藏。
- wx:if 有更高的切换消耗而 hidden 有更高的初始渲染消耗。因此，如果需要频繁切换时，用 hidden 更好；如果在运行时，条件不大可能改变，则 wx:if 更好。

4）模板

WXML 提供模板（template），可以在模板中定义代码片段，然后在不同的地方调用。使用 name 属性作为模板的名字，然后在<template/>内定义代码片段，代码如下所示：

```
<!--template.wxml-->
<template name="StudentName">
  <view>
    FirstName: {{firstName}}, LastName: {{lastName}}
  </view>
</template>
```

使用 is 属性声明需要使用的模板，然后将模板所需要的 data 传入，模板拥有自己的作用域，只能使用 data 传入的数据，代码如下所示：

```
<!--template.wxml-->
<!--使用模板-->
<template is="StudentName" data="{{...StudentA}}"></template>
<template is="StudentName" data="{{...StudentB}}"></template>
<template is="StudentName" data="{{...StudentC}}"></template>

// template.js
Page({
    data: {
        StudentA: { firstName: 'San', lastName: 'Zhang' },
        StudentB: { firstName: 'Si', lastName: 'Li' },
        StudentC: { firstName: 'Mingyang', lastName: 'Liu' }
    }
})
```

5）事件绑定

事件绑定在组件上，当达到触发事件，就会执行逻辑层中对应的事件处理函数。将用户的行为反馈到逻辑层进行处理，是视图层到逻辑层的通信方式。事件对象可以携带额外信息，如

id、dataset、touches。

组件绑定事件，当用户单击该组件时会在该页面对应的 Page 中找到相应的事件处理函数。代码如下所示：

```
<!--eventBinding.wxml-->
<view bindtap="add"> {{count}} </view>
<button bindtap="addTest"> 累加 </button>

// eventBinding.js
Page({
    data: {
        count: 1
    },
    add: function (e) {
        this.setData({
            count: this.data.count + 1
        })
    },
    addTest: function (e) {
        this.setData({
            count: this.data.count + 1
        })
    }
})
```

在相应的Page定义中写上相应的事件处理函数，参数是e。Log出来的信息如图2-24所示：

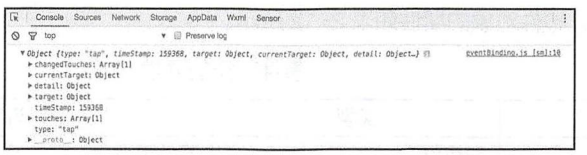


图 2-24 事件参数信息

WXML 的事件分为冒泡事件和非冒泡事件。冒泡事件：当一个组件上的事件被触发后，该事件会向父节点传递。非冒泡事件：当一个组件上的事件被触发后，该事件不会向父节点传递。

WXML 的冒泡事件类型如表 2-11 所示。

表 2-11 WXML 的冒泡事件类型

类型	触发条件
touchstart	手指触摸
touchmove	手指触摸后移动
touchcancel	手指触摸动作被打断，如来电提醒、弹窗
touchend	手指触摸动作结束
tap	单击事件（手指触摸后离开）
longtap	长按事件（手指触摸后，超过 350ms 再离开）

除表 2-11 之外的其他组件自定义事件都是非冒泡事件，如<form/>的 submit 事件、

<input/>的 input 事件、<scroll-view/>的 scroll 事件等。

事件绑定的写法和组件的属性一样，使用 key=value 的形式。key 以 bind 或 catch 开头，然后跟上事件的类型，如 bindtap、catchtouchstart 等，value 是一个字符串，对应 Page 中定义的函数名字。value 与函数一一对应。

bind 事件绑定不会阻止冒泡事件向上冒泡，catch 事件绑定可以阻止冒泡事件向上冒泡。例如下面的例子，单击 Sun view 会先触发 tap3 事件，向上传递继续触发 tap2 事件，因为 tap2 事件使用的是 catch 类型事件，所以会阻止事件继续向上传递，不能执行 tap1 事件，只有单击 Parent view 才可以执行自己的 tap1 事件。

具体代码如下所示：

```
<!--eventBinding.wxml-->
<view id="outter" bindtap="tap1">
  Parent view
  <view id="middle" catchtap="tap2">
    Sub view
    <view id="inner" bindtap="tap3">
      Sun view
    </view>
  </view>
</view>
```

当组件触发事件时，逻辑层绑定该事件的处理函数会收到一个事件对象，从事件对象中可以获取对象的属性。事件对象的属性如表 2-12 所示。

表 2-12　　事件对象的属性

属性	类型	说明
type	String	事件类型
timeStamp	Integer	事件生成时的时间戳
target	Object	触发事件的组件的一些属性值集合
currentTarget	Object	当前组件的一些属性值集合
touches	Array	触摸事件，触摸点信息的数组
detail	Object	额外的信息

target 和 currentTarget 可以参考上例，单击 Sun view 时，tap3 收到的事件对象 target 和 currentTarget 都是 Sun view，而 tap2 收到的事件对象 target 是 Sun view，currentTarget 是 Sub view。

<canvas/> 中的触摸事件属于特殊事件，不可冒泡，所以没有 currentTarget。自定义事件 CustomEvent 对象属性（继承 BaseEvent）如表 2-13 所示。

表 2-13　　CustomEvent 对象属性

属性	类型	说明
detail	Object	额外的信息

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

TouchEvent 触摸事件对象属性（继承 BaseEvent）如表 2-14 所示。

表 2-14 TouchEvent 触摸事件对象属性

属性	类型	说明
touches	Array	触摸事件，当前停留在屏幕中的触摸点信息的数组
changedTouches	Array	触摸事件，当前变化的触摸点信息的数组

关于表 2-12～表 2-14 属性的几点说明：

- type：通用事件类型。
- timestamp：从该页面打开到触发事件所经过的毫秒数。
- target：触发事件的源组件，包括以下属性，如表 2-15 所示。

表 2-15 target 的属性

属性	说明
id	事件源组件的 id
dataset	事件源组件上由 data-开头的自定义属性组成的集合
offsetLeft、offsetTop	事件源组件在坐标系统中的偏移量

dataset：在组件中可以定义数据，这些数据将会通过事件传递给 SERVICE。书写方式：以 data-开头，多个单词由连字符“-”连接，不能有大写（大写会自动转成小写），如 data-element-type，最终在 event.target.dataset 中会将连字符转成驼峰 elementType。代码如下所示：

```
    <!--eventBinding.wxml-->
    <view data-alpha-beta="1" data-alphaBeta="2" bindtap="bindViewTap"> DataSet
Test </view>

    // eventBinding.js
       bindViewTap:function(event){
       event.target.dataset.alphaBeta === 1 // - 会转为驼峰写法
       event.target.dataset.alphabeta === 2 // 大写会转为小写
     }
```

- currentTarget：事件绑定的当前组件，包括以下属性，如表 2-16 所示。

表 2-16 currentTarget 的属性

属性	类型	说明
id	String	当前组件的 id
tagName	String	当前组件的类型
dataset	Object	当前组件上由 data-开头的自定义属性组成的集合

- touches：是一个数组，每个元素为一个 Touch 对象（canvas 触摸事件中携带的 touches 是 CanvasTouch 数组）。touches 表示当前停留在屏幕上的触摸点。Touch 每个触摸点对象有以下属性，如表 2-17 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 2-17 Touch 每个触摸点的属性

属性	类型	说明
identifier	Number	触摸点的标识符
pageX、pageY	Number	距离文档左上角的距离，文档的左上角为原点，横向为 X 轴，纵向为 Y 轴
clientX、clientY	Number	距离页面可显示区域（屏幕除去导航条）左上角距离，横向为 X 轴，纵向为 Y 轴

CanvasTouch 对象的属性如表 2-18 所示。

表 2-18 CanvasTouch 对象的属性

属性	类型	说明
identifier	Number	触摸点的标识符
x、y	Number	距离 Canvas 左上角的距离，Canvas 的左上角为原点，横向为 X 轴，纵向为 Y 轴

- changedTouches：数据格式同 touches，表示有变化的触摸点，如从无变有（touchstart）、位置变化（touchmove）、从有变无（touchend、touchcancel）。
- detail：自定义事件所携带的数据，如表单组件的提交事件会携带用户的输入信息，媒体的错误事件会携带错误信息。

6）引用

WXML 提供两种文件引用方式 import 和 include。

import 可以在该文件中使用目标文件定义的 template，代码如下所示：

在 a.wxml 中定义了一个叫 A 的 template：

```
<!--pages/a/a.wxml-->
<template name="A">
  <text>{{text}}</text>
</template>
```

在 b.wxml 中引用了 a.wxml，就可以使用 A 模板：

```
<!--pages/b/b.wxml-->
<import src="../../pages/a/a.wxml"/>
<template is="A" data="{{text: 'b import a'}}"/>
```

import 有作用域的概念，即只会 import 目标文件中定义的 template，而不会 import 目标文件 import 的 template。c import b，b import a，在 c 中可以使用 b 定义的 template，在 b 中可以使用 a 定义的 template，但是 c 不能使用 a 定义的 template。代码如下所示：

```
<!--pages/a/a.wxml-->
<template name="A">
  <text>{{text}}</text>
</template>

<!--pages/b/b.wxml-->
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
<import src="../../pages/a/a.wxml"/>
<template is="A" data="{{text: 'b import a'}}"/>

<template name="B">
  <text>{{text}}</text>
</template>

<!--pages/c/c.wxml-->
<import src="../../pages/b/b.wxml"/>
<template is="B" data="{{text: 'c import b'}}"/>
<template is="A" data="{{text: 'c import a'}}"/>

<template name="C">
  <text>{{text}}</text>
</template>
```

调试模式下运行，会看到 Console 面板输出错误信息，如图 2-25 所示。

include 可以将目标文件中除了<template/>的整个代码引入，相当于复制到 include 位置，例如 c.wxml 文件 include a.wxml 和 b.wxml 文件，则 c.wxml 文件会显示 a.wxml 和 b.wxml 文件的内容。代码如下所示：

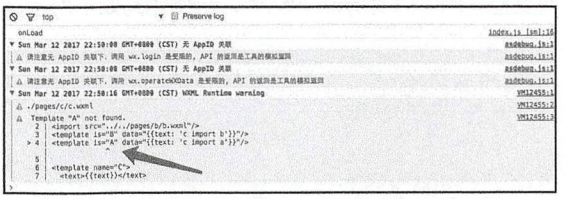


图 2-25 Console 面板

```
<!--pages/c/c.wxml-->

<view>**********</view>
<view> c.wxml 开始</view>
<include src="../../pages/a/a.wxml"/>
<include src="../../pages/b/b.wxml"/>
<view> c.wxml 结束</view>
```

2.6 视图层 WXSS 介绍

WXSS（WeiXin Style Sheets）是 MINA 框架设计的一套样式语言，用于描述 WXML 的组件样式。

WXSS 用来决定 WXML 的组件应该怎么显示。为了适应广大的前端开发者，微信 WXSS 具有 CSS 大部分特性。同时为了更适合开发微信小程序，微信对 CSS 进行了扩充以及修改。与 CSS 相比扩展的特性有：尺寸单位、样式导入。

1）尺寸单位

WXSS 拥有针对屏幕的两种尺寸单位：rpx、rem。

- rpx（responsive pixel）：可以根据屏幕宽度进行自适应，规定屏幕宽为 750rpx。如在 iPhone6 上，屏幕宽度为 375px，共有 750 个物理像素，则 750rpx = 375px = 750 物理像素，1rpx = 0.5px = 1 物理像素。rpx 与 px 的换算如表 2-19 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 2-19 rpx 与 px 的换算

设备	rpx 换算 px（屏幕宽度/750）	px 换算 rpx（750/屏幕宽度）
iPhone5	1rpx = 0.42px	1px = 2.34rpx
iPhone6	1rpx = 0.5px	1px = 2rpx
iPhone6s	1rpx = 0.552px	1px = 1.81rpx

● rem（root em）：规定屏幕宽度为 20rem；1rem = (750/20)rpx。因此建议：开发微信小程序时设计师可以用 iPhone6 作为视觉稿的标准。在较小的屏幕上不可避免会有一些毛刺，请在开发时尽量避免这种情况。

2）样式导入

使用@import 语句可以导入外联样式表，@import 语句后为需要导入的外联样式表的相对路径，用“;”表示语句结束。示例代码如下所示：

```
/** common.wxss **/
.small-p{
  padding:5px;
}

/** app.wxss **/
@import "common.wxss";
.middle-p:{
  padding:15px;
}
```

内联样式

MINA 组件上支持使用 style、class 属性来控制组件的样式。

● style：静态的样式统一写到 class 中。style 接收动态的样式，在运行时会进行解析，所以不要将静态的样式写进 style 中，以免影响渲染速度。使用代码如下所示：

```
<view style="color:{{color}};" />
```

● class：用于指定样式规则，其属性值是样式规则中类选择器名（样式类名）的集合，样式类名不需要带上“.”，样式类名之间用空格分隔。比如 normal_view 样式类的使用代码如下所示：

```
<view class="normal_view" />
```

全局样式与局部样式

在 app.wxss 中定义的样式为全局样式，作用于每一个页面。在 page 的 wxss 文件中定义的样式为局部样式，只作用在对应的页面，并会覆盖 app.wxss 中相同的选择器。

WXSS 目前支持的选择器如表 2-20 所示。

表 2-20 WXSS 目前支持的选择器

选择器	样例	样例描述
.class	.intro	选择所有拥有 class="intro"的组件
#id	#firstname	选择拥有 id="firstname"的组件

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

续表

选择器	样例	样例描述
Element	view	选择所有 view 组件
element, element	view checkbox	选择所有文档的 view 组件和所有的 checkbox 组件
::after	view::after	在 view 组件后边插入内容
::before	view::before	在 view 组件前边插入内容

2.7 WXML 与 HTML 的区别

HTML 是用于创建网页的语言。开发者通过使用 HTML 标记标签创建 html 文档来创建网页。HTML 代表超文本标记语言。HTML 是一种标记语言，它具有标记标签的集合，其标签数量有 80 多个。

HTML 标签是由尖括号（如<html>、<body>）包围的字词。标签通常成对出现，例如<html>和</html>。

一对中的第一个标签是开始标签；第二个标签是结束标签。在上面的示例中，<html>是开始标签，而</html>是结束标签。可以将开始标签称为起始标签，结束标签称为闭合标签。

WXML 虽然也是类似于 HTML 语言的起始标签和闭合标签，但是 WXML 提供的标签很有限。WXML 只是通过 8 大类组件来模拟 HTML 各种标签的使用。两者标签的差异如表 2-21 所示。

表 2-21 WXML 与 HTML 标签对比

WXML	HTML
view	article、aside、bady、div、 ul、li、caption、dd、dl、dt、 footer、header、nav、section、 table、thead、tbody、tr、td、th、ol、 h1、h2、h3、h4、h5、h6、p、em
icon	.icon 后缀格式的图片文件，存放在根目录或在<link>中使用
text	span
button	button
checkbox	input[type = ‘checkbox’]
form	form
input	input[type = ‘text’]
image	img
radio	input[type = ‘radio’]
navigator	a

可以看出 WXML 组件很少。关于 WXML 组件标签详细介绍，请参考第三章，这里有几点说明：

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

- <view>组件代替了 HTML 中大部分标签。
- 只有<text>标签可以长按选中文本，<text>组件不允许嵌套使用，支持转义符号“/”。
- <navigator>目前只能在小程序内部跳转导航，不能跳转到外部 url。

2.8 小程序调试、上传、发布

第一章已经介绍了如何申请开发者账号和获取小程序的 AppID，开发完成小程序，测试没有问题，就可以上传发布上线。本节主要介绍小程序的预览和发布流程。

2.8.1 事前准备：HTTPS

为了保护小程序应用安全，微信官方的需求文档要求，每个小程序必须事先设置一个通信域名，并通过 HTTPS 请求进行网络通信，不满足条件的域名和协议无法请求，如图 2-26 所示。因此开发者应先准备好 HTTPS 证书的域名，配置好 HTTPS。

域名要实现 HTTPS 加密请求，需要安装 SSL 证书。SSL 证书的类型有以下几种：

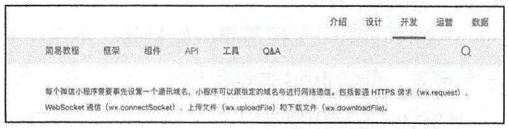


图 2-26 微信官方文档内容

- DV SSL 证书（域名验证型）：只验证域名所有权，适合个人网站、博客等站点使用。
- IV SSL 证书（个人验证型）：验证网站所属个人身份，适合自媒体、个人品牌站点使用。
- OV SSL 证书（企业验证型）：验证网站所属单位身份，适合企业级用户使用。
- EV SSL 证书（扩展验证型）：扩展验证网站所属单位身份，适合高度信任的企业级用户使用。

以上 4 种类型的 SSL 证书加密强度如表 2-22 所示。

表 2-22 SSL 证书加密强度

证书类型	强制加密	单域名 www.domain.com	通配域名 *.domain.com	多域名 www.domain.com www.domainA.com	混合域名 *.domain.com www.domainA.com
DV SSL 证书	不支持	支持	不支持	支持	不支持
IV SSL 证书	不支持	支持	支持	支持	支持
OV SSL 证书	支持	支持	支持	支持	支持
EV SSL 证书	支持	支持	不支持	支持	不支持

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

正式上线后网络请求的域名是申请时填写的域名，而且必须是 https 的。登录微信公众平台-小程序后台，在“设置”-“开发设置”下面可以配置服务器域名，如图 2-27 所示。

这里需要管理员扫码才可以进行配置，配置界面如图 2-28 所示。

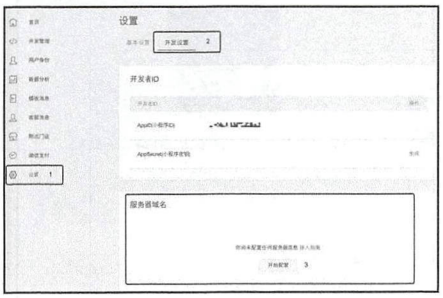


图 2-27 服务器域名

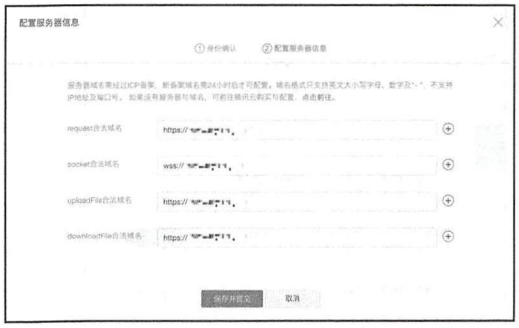


图 2-28 配置服务器域名

服务器域名配置好之后，一个月内可以申请 5 次修改，如图 2-29 所示。

开发者可以在微信开发者工具，“项目”-“配置信息”中看到相关信息，如图 2-30 所示。

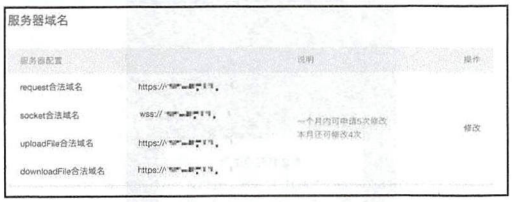


图 2-29 域名信息

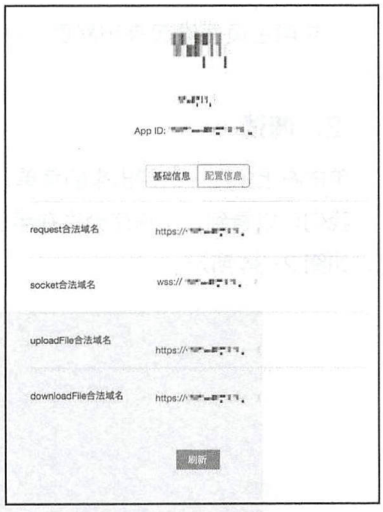


图 2-30 项目配置信息

2.8.2 预览及调试

1. 预览

开发完成之后，除了模拟器测试外，为了查看有没有问题，更真实地去体验小程序，开发者都会选择在手机上预览，预览测试完毕才会发布。在开发者工具中选择“项目”，再单击“预览”，如图 2-31 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

AppID 没有问题的话，先进行打包，打包之后代码非常小，然后上传，上传完成之后，会弹出一个二维码。开发者需要扫码预览，如图 2-32 所示。扫码之后即可在手机看到小应用。

图 2-31 项目界面

图 2-32 扫码预览

小应用主页面看起来和网页 H5 应用差别不大，主要还是在体验上反应速度更快、更流畅。

2. 调试

单击右上角“…”弹出来的菜单，如图 2-33 所示。

我们可以看到，小程序允许在手机上调试，单击“打开调试”。第一次打开会看到提示信息，如图 2-34 所示。

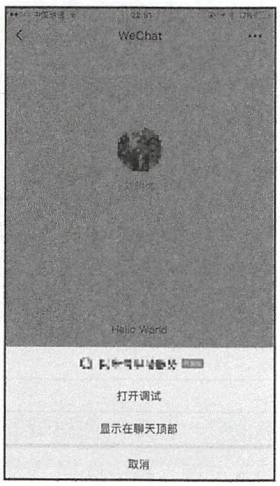


图 2-33 菜单选项

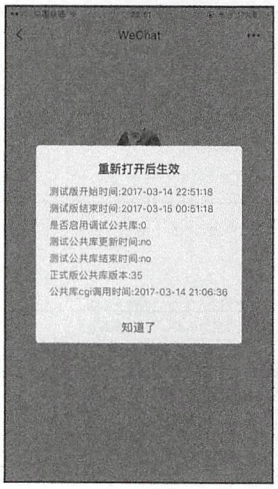


图 2-34 打开调试提示

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

重新预览小程序，可以看到右下角显示一个绿色“vConsole”按钮，如图 2-35 所示。单击“vConsole”按钮，可以进入控制台，如图 2-36 所示。

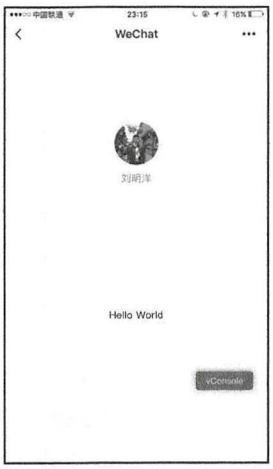


图 2-35 “vConsole”按钮

图 2-36 控制台

这里主要是查看系统及网络相关的一些信息，开发者在调试的时候，打出来的日志可以在第一个 Log 下查看，因为小程序不能弹出提示框来显示数据，所以通过控制台输出会很方便查看一些数据。

2.8.3 发布

1. 上传

测试完毕之后，确认没问题，就可以单击图 2-37 中的“上传”按钮，上传到微信后台，这里只有管理员才能进行操作，如图 2-37 所示。

单击上传按钮，只有管理员的开发工具才拥有上传代码权限，扫码通过之后，需要填写小程序版本号、项目备注信息，如图 2-38 所示。

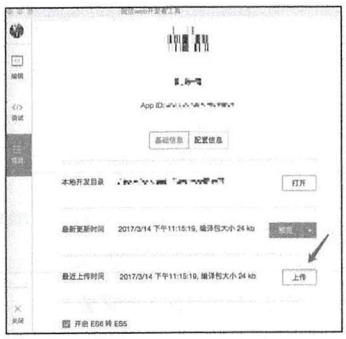


图 2-37 上传

图 2-38 上传小程序到微信平台

非卖品，仅供非商业用途或交流学习使用

2. 提交前的准备

登录微信小程序的后台，在提交审核之前需要先确定完成以下几步，如图 2-39 所示。

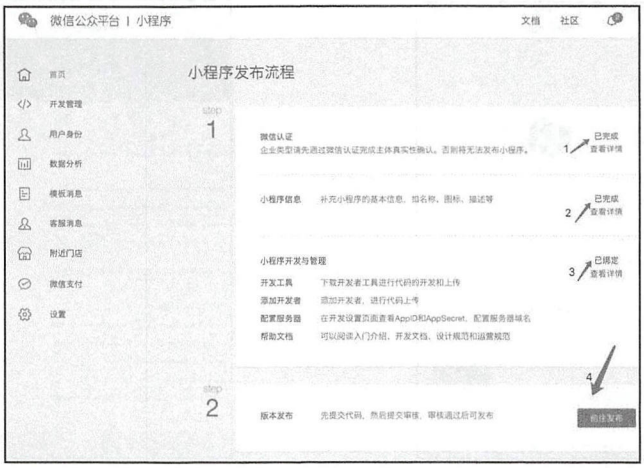


图 2-39 小程序发布流程

（1）微信认证企业类型请先通过微信认证完成主体真实性确认，否则将无法发布小程序。

（2）补充小程序的基本信息，如名称、图标、描述等。

（3）在小程序开发与管理中可以添加开发者，配置小程序服务器域名等信息。

（4）前往发布，提交审核。

3. 提交审核

前面的准备工作做完后，登录微信公众平台小程序后台，单击左侧的“开发管理”菜单，“开发版本”中展示已上传的代码及版本号，如图 2-40 所示，管理员可以提交审核、选为体验版本或是删除项目，如图 2-41 所示。

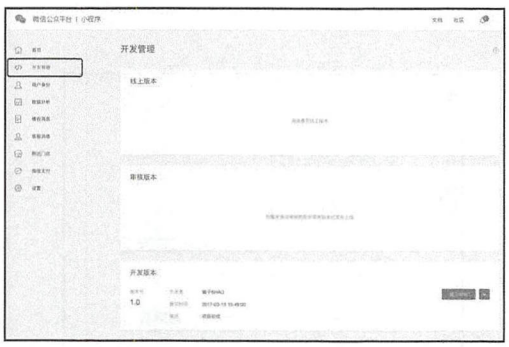


图 2-40 开发版本

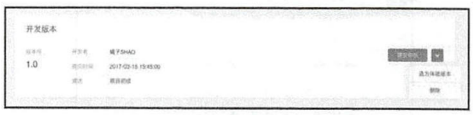


图 2-41 开发版本设置

单击“选为体验版”，管理员扫码允许之后，可以生成体验版二维码，如图 2-42 所示。

下载体验版二维码，可以分发给管理员或体验者，扫码使用小程序，体验者可以通过小程序后台“用户身份”下面“体验者”进行添加，如图 2-43 所示。

图 2-42 体验版二维码

图 2-43 体验者管理

单击体验者，管理员扫码允许之后，可以通过微信号搜索用户，绑定成为体验者，最多可以绑定 40 位体验者，如图 2-44 所示。

在“开发版本”中单击“提交审核”，会提示提交审核的相关须知，如图 2-45 所示。

图 2-44 绑定体验者

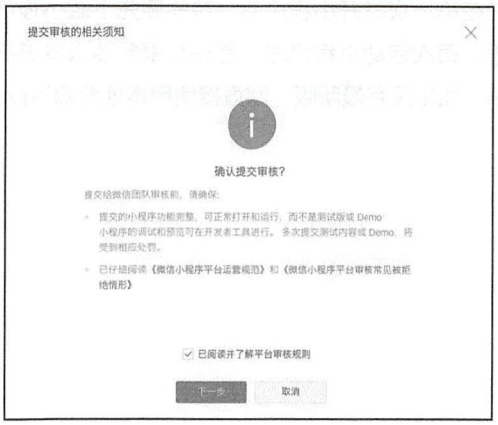


图 2-45 提交审核的相关须知

单击“下一步”之后，出现二维码，需要管理员扫码允许，允许之后，需要填写审核信息，如图 2-46 所示。

为了用户可以快速搜索出小程序，在这里可以填写一些重要的页面类目与标签，最多添加 5 组。单击“提交审核”之后，可以在“开发管理”下的“审核版本”中看到，如图 2-47 所示。

由于大小限制，如果代码包大小超过 2048KB，提交审核时开发工具会提示代码超出大小限制的提示。提交成功之后，可以在“审核版本”栏目中看到审核的进度。另外需要注意

的是，如果审核没有通过，可以参考“微信小程序平台常见拒绝情形”进行分析修改后重新提交审核。

图 2-46　填写审核信息

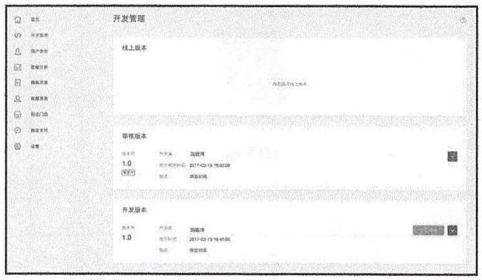


图 2-47　审核版本

4. 发布

审核通过后，需要管理员手动单击“发布”，小程序才会发布到线上供用户搜索使用。用户在第一次打开小程序时，首先要先下载小程序所有资源，下载成功之后，即可在微信中使用。每次启动小程序时，首先与服务器版本进行对比，如果有最新版，则继续下载最新版资源；如果没有最新版，则直接使用本地资源运行。

第三章

框架组件

MINA 框架提供了一系列基础组件，通过组合这些基础组件进行快速开发。小程序提供了一些标准的 View，大大节省了 UI 开发成本，这也使得小程序风格一致，体验流畅。小程序除了提供一些基础组件，如文本（text）、图标（icon）、进度条（progress）、滚动条（scroll-view）、表单组件（button、checkbox、checkbox-group、form、input、label、picker、picker-view、radio、slider、switch、textarea）等，还提供了媒体组件、地图和画布功能组件。

3.1 框架组件介绍

MINA 框架为开发者提供了一系列基础组件，开发者可以通过组合这些基础组件进行快速开发。所谓组件就是视图层的基本组成单元，自带一些功能与微信风格的样式。一个组件通常包括开始标签和结束标签，通过属性来修饰这个组件，内容在两个标签之内，所有组件与属性都是小写的，单词之间用连字符“-”连接。代码如下所示：

```
<view class="usermotto">
    <text class="user-motto">{{motto}}</text>
</view>
```

每个组件可以自定义属性，用于组件的功能或样式展示，但属性只支持下面 7 种数据类型，如表 3-1 所示。

表 3-1　　数据类型

类型	描述	注解
Boolean	布尔值	组件写上该属性，不管该属性等于什么，其值都为 true，只有组件上没有写该属性时，属性值才为 false。如果属性值为变量，变量的值会被转换为 Boolean 类型
Number	数字	如 1、2.5
String	字符串	如"string"
Array	数组	如[1, "string"]
Object	对象	如{ key: value }

续表

类型	描述	注解
EventHandler	事件处理函数名	"handlerName" 是 Page 中定义的事件处理函数名
Any	任意属性	适用于任意属性数据类型

所有组件都有公共的属性，如表 3-2 所示。

表 3-2　　公共的属性

属性名	类型	描述	注解
id	String	组件的唯一标识	保持整个页面唯一
class	String	组件的样式类	在对应的 wxss 中定义的样式类
style	String	组件的内联样式	可以动态设置的内联样式
hidden	Boolean	组件是否显示	所有组件默认显示
data-*	Any	自定义属性	组件上触发事件时，会发送给事件处理函数
bind* / catch*	EventHandler	组件的事件	详见事件

另外，几乎所有组件都有各自定义的属性，可以对该组件的功能或样式进行修饰，请参考各个组件的定义。微信小程序提供了 10 大类组件，组件列表如表 3-3 所示。

表 3-3　　组件

视图容器（view Container）	
组件名	说明
view	视图容器
scroll-view	可滚动的视图容器
swiper	可滑动的视图容器
movable-area	可移动的视图容器，在页面中可以拖曳滑动
cover-view	覆盖在原生组件之上的文本视图
基础内容（Basic Content）	
组件名	说明
icon	图标
text	文字
rich-text	富文本
progress	进度条
表单（Form）	
组件名	说明
button	按钮
form	表单

续表

组件名	说明
input	输入框
checkbox	单项选择器
radio	多项选择器
picker	列表选择器
picker-view	嵌入页面的滚动选择器
slider	滑动选择器
switch	开关选择器
label	标签
textarea	多行输入框
操作反馈（Interaction）	
组件名	**说明**
action-sheet	上拉菜单
modal	模态弹窗
toast	短通知
导航（Navigation）	
组件名	**说明**
navigator	应用内跳转
多媒体（Media）	
组件名	**说明**
audio	音频
image	图片
video	视频
camera	系统相机
地图（Map）	
组件名	**说明**
map	地图
画布（Canvas）	
组件名	**说明**
canvas	画布
开放数据（open-data）	
组件名	**说明**
open-data	开放数据

续表

客服会话按钮（contact-button）	
组件名	说明
contact-button	客服会话按钮

其中，视图容器、基础内容、表单、操作反馈、导航这 5 大组件，称为“常用组件”，开发者经常会用到。多媒体组件、地图组件和画布组件称为“高级组件”，使用场景相对比较复杂。

3.2 视图容器

视图容器组件有 5 个：view、scroll-view、swiper 与 swiper-item、movable-area 与 movable-view、cover-view 与 cover-image。主要用于界面布局和展示。

3.2.1 view

view 组件相当于 html 中的 标签，有 4 个属性，如表 3-4 所示。

表 3-4 view 组件的属性

属性名	类型	默认值	说明
hover	Boolean	false	是否启用单击效果
hover-class	String	none	指定按下去的样式类。当 hover-class="none" 时，没有单击效果
hover-start-time	Number	50	按住后多久出现单击效果，单位为 ms
hover-stay-time	Number	400	手指松开后单击效果持续时间，单位为 ms

hover 和 hover-class 与单击效果有关，分别设置是否启用单击效果和设置单击的效果。如果 view 没有 hover 属性，代表 hover=" false "。

hover-start-time 和 hover-stay-time 与单击效果的时间有关，分别设置延迟多久才出现单击的效果和单击效果持续的时间，单位都是 ms。如果 view 没有 hover-start-time 和 hover-stay-time 属性，在 hover="true"的状态下，延迟时间默认为 50ms，持续时间默认为 400ms。

在以下代码中设置 3 个 view：

1）第 1 个 view：hover="true"，开启单击效果。hover-start-time="1000"代表 1000ms 之后才出现单击效果。没有 hover-stay-time 属性，则代表持续时间使用默认值 400ms。

2）第 2 个 view：hover="true"，开启单击效果。hover-stay-time="3000"代表单击效果持续 3000ms。没有 hover-start-time 属性，则代表延迟出现单击效果的时间使用默认值 50ms。

3）没有 hover 属性，代表 hover=" false "，没有单击效果。

具体代码如下所示，运行效果如图 3-1 所示。

```
<!--pages/viewPage1/viewPage1.wxml-->
<view class="container">
    <view class="flex-item class_red" hover="true" hover-class="r_hover"
hover-start-time="1000">按下 1000ms 后才出现单击效果</view>
    <view class="flex-item class_green" hover="true" hover-class="g_hover"
hover-stay-time="3000">单击保留 3000ms</view>
    <view class="flex-item class_blue">没有单击效果</view>
</view>

/* pages/viewPage1/viewPage1.wxss */
.flex-item{
  width: 100%;
  height: 100px;
  box-sizing: border-box;
}
.class_red{
  background-color: #ff7256;
}
.class_green{
  background-color: #7CCD7C;
}
.class_blue{
  background-color: #6495ED;
}
.g_hover{
  border: 6px solid #CD8500;
}
.r_hover{
  border: 6px solid #CD8500;
}
```

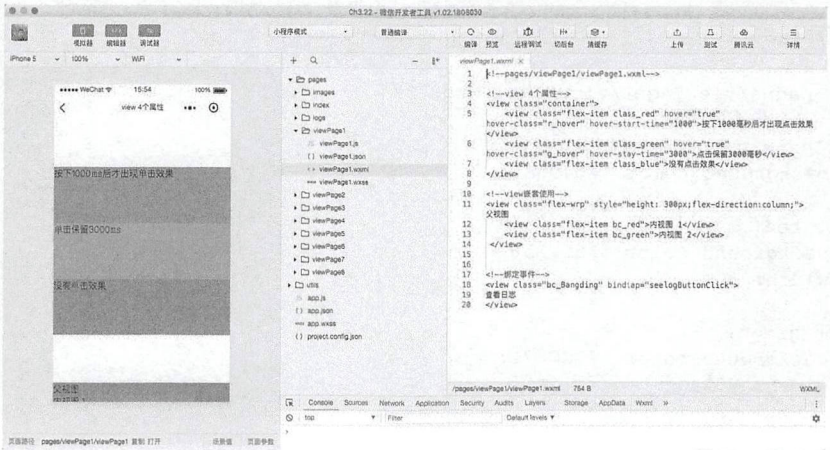


图 3-1 view 3 个属性的运行效果

另外，从上面代码可以看到 viewPage1.wxml 文件中 view 都有 class 属性，例如第一个 view 属性 class="flex-item class_red"，其代表 view 使用 flex-item 和 class_red 两个样

式，这两个样式存在.wxss 文件中，也就是我们之前常说的 CSS 样式。

```
.flex-item{
  width: 100%;
  height: 100px;
  box-sizing: border-box;
}
.class_red{
  background-color: #ff7256;
}
```

- width：100%代表组件宽度是屏幕宽度的 100%，也就是和屏幕宽度一样。
- height：100px 代表高度是 100 像素。
- box-sizing 属性可以为 3 个值之一：content-box（default）、border-box、padding-box。

content-box：border 和 padding 不计算入 width 之内。

border-box：border 和 padding 计算入 width 之内。

padding-box：padding 计算入 width 内。

- background-color：#ff7256 代表组件背景颜色值是#ff7256。

view 视图组件可以单个使用，也可以嵌套使用，给一个 view 组件上添加几个 view 内视图，代码如下所示：

```
<!--pages/viewPage1/viewPage1.wxml-->
<!--view 嵌套使用-->
<view class="flex-wrp" style="height: 300px;flex-direction:column;"> 父视图
    <view class="flex-item bc_red">内视图 1</view>
    <view class="flex-item bc_green">内视图 2</view>
 </view>

/* pages/viewPage1/viewPage1.wxss */
/*view 嵌套使用*/
.flex-wrp{
  background-color: #6495ED;
}
.bc_red{
  background-color: #ff7256;
  width: 80%;
}
.bc_green{
  background-color: #7CCD7C;
  width: 80%;
}
```

运行效果，如图 3-2 所示。

使用 bindtap 属性进行绑定事件，bindtap 的值等于事件的名称，代码如下所示：

```
<!--pages/viewPage1/viewPage1.wxml-->
<!--绑定事件-->
<view class="bc_Bangding" bindtap="seelogButtonClick">
```

```
查看日志
</view>

/* pages/viewPage1/viewPage1.wxss */
/*绑定事件*/
.bc_Bangding{
  background-color: #CD3278;
  width: 100%;
  height: 100px;
}

// pages/viewPage1/viewPage1.js
Page({
  data:{},
  seelogButtonClick:function(event){
    console.log(event)
    wx.navigateTo({
      url: '../logs/logs'
    })
  }
})
```

上面的代码通过 bindtap="seelogButtonClick" 绑定 seelogButtonClick 事件，seelogButtonClick 写在.js 文件内容 page 下面。单击 view 会触发 seelogButtonClick 事件，跳到一个名为 logs 的 Page。

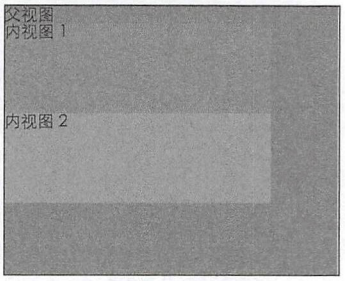


图 3-2 view 嵌套使用

微信小程序页面布局方式采用的是 Flex 布局，下面介绍一下 Flex 布局在微信小程序中的使用。Flex 是 Flexible Box 的缩写，意为“弹性布局”，可以简便、完整地实现各种页面布局。Flex 布局提供了元素在容器中的对齐、方向以及顺序，它们甚至可以是动态的或者不确定大小的，能够调整其子元素在不同大小的屏幕中用最适合的方法填充合适的空间。

Flex 布局的特点：

- 可任意方向伸缩，向左、向右、向下、向上。
- 在样式层可以调换和重排顺序。
- 主轴和侧轴方便配置。
- 子元素的空间拉伸和填充。
- 沿着容器对齐。

小程序通过 display 来设置显示的模式。一共有两种显示模式，分别是 display:block 和 display:flex。

- display:block：块内容器模式，总是使用新行开始显示，视图容器（view、scroll-

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

view 和 swiper）默认都是 dispaly:block 显示模式。例如添加两个 view 时，会一上一下，显示两行。

- display:flex：行内容器模式，在一行内显示子元素，可以使用 flex-wrap 属性指定其是否换行，flex-wrap 有 3 个值：nowrap（不换行）、wrap（换行）、wrap-reverse（换行，第一行在下面）。

下面分别使用 display:block 和 display:flex 演示 view 的布局效果，代码如下所示。显示效果如图 3-3 所示。

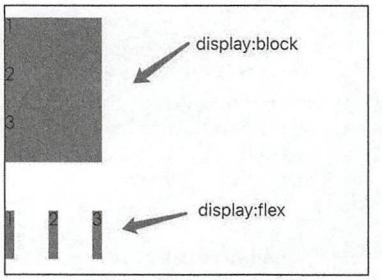


图 3-3 display 两种显示模式

```
<!--pages/viewPage2/viewPage2.wxml-->

<!--display:block 的显示-->
<view class="flex-row1" style="display: block;">
  <view class="flex-view-item1">1</view>
  <view class="flex-view-item1">2</view>
  <view class="flex-view-item1">3</view>
</view>

<!--display:flex 的显示-->
<view class="flex-row" style="display: flex;">
  <view class="flex-view-item">1</view>
  <view class="flex-view-item">2</view>
  <view class="flex-view-item">3</view>
</view>

/* pages/viewPage2/viewPage2.wxss */
.flex-row {
  width: 100px;
  height: 150px;
  flex-direction: row;
  justify-content: space-between;
  align-items: center;
}

.flex-view-item {
  height: 50px;
  background-color: #ff5400;
}
```

从效果图可以看出 block 和 flex 两种容器的区别，block 模式下子元素 view 是换行的，flex 模式下是子元素 view 显示在一行内。

采用 flex 布局的元素，称为 flex 容器（flex container），简称“容器”，上面定义的 view 都是容器。它的所有子元素自动成为容器成员，称为 flex 项目（flex item），简称“项目”。flex 容器的架构图，如图 3-4 所示。

容器默认有两个轴：主轴（main axis）和侧轴（cross axis）。主轴的开始位置为主轴起点（main start），结束位置为主轴终点（main end），而主轴的长度为主轴长度（main

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

size）。同理，侧轴的开始位置为侧轴起点（cross start），结束位置为侧轴终点（cross end），长度为侧轴长度（cross size）。flex item（项目）默认沿主轴排列。单个项目占据的主轴空间叫作 main size，占据的交叉轴空间叫作 cross size。

flex container（容器）的属性

1）flex-direction 属性

图 3-4 只是演示容器的结构（主轴和侧轴），但是需要注意主轴并不一定是从左到右的，同理侧轴也不一定是从上到下的，如果水平方向为主轴，那么垂直方向就是侧轴，反之亦然。主轴的方向使用 flex-direction 属性控制，它有 4 个可选值：

- row：从左到右的水平方向为主轴。
- row-reverse：从右到左的水平方向为主轴。
- column：从上到下的垂直方向为主轴。
- column-reverse：从下到上的垂直方向为主轴。

flex-direction 属性 4 个值使用的代码如下所示，对应的 4 种主轴方向设置的效果如图 3-5 所示。

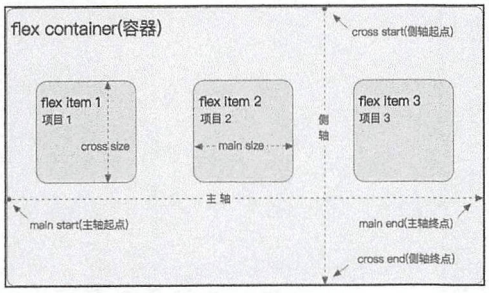


图 3-4 flex 容器的架构图

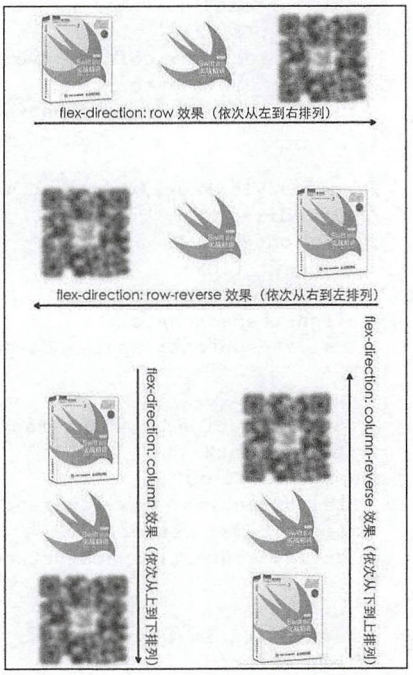


图 3-5 flex-direction 属性对应的 4 种主轴方向设置效果

```
/* pages/viewPage2/viewPage2.wxss */
/*flex-direction: row 效果*/
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
.view-content1 {
  display: flex;
  flex-direction: row;
  align-items: center;
  justify-content: space-between;
}

/*flex-direction: row-reverse 效果*/
.view-content2 {
  display: flex;
  flex-direction: row-reverse;
  align-items: center;
  justify-content: space-between;
}

/*flex-direction: column 效果*/
.view-content3 {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: space-between;
}

/*flex-direction: column-reverse 效果*/
.view-content4 {
  display: flex;
  flex-direction: column-reverse;
  align-items: center;
  justify-content: space-between;
}

/* pages/viewPage2/viewPage2.wxss */
/*flex-direction: row 效果*/
.view-content1 {
  display: flex;
  flex-direction: row;
  align-items: center;
  justify-content: space-between;
}

/*flex-direction: row-reverse 效果*/
.view-content2 {
  display: flex;
  flex-direction: row-reverse;
  align-items: center;
  justify-content: space-between;
}

/*flex-direction: column 效果*/
.view-content3 {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: space-between;
}
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
/*flex-direction: column-reverse 效果*/
.view-content4 {
  display: flex;
  flex-direction: column-reverse;
  align-items: center;
  justify-content: space-between;
}
```

2）flex-wrap 属性

当子元素都排在一条线（“轴线”）上时，flex-wrap 属性定义，如果一条轴线排不下，如何换行。该属性有如下 3 种情况：

wrap：表示一行排不下，就换行。

nowrap：表示即使一行排不下，也不换行。

wrap-reverse：换行，第一行在下方。

flex-wrap 属性的 3 个值使用代码如下所示，对应的 3 种布局效果如图 3-6 所示。

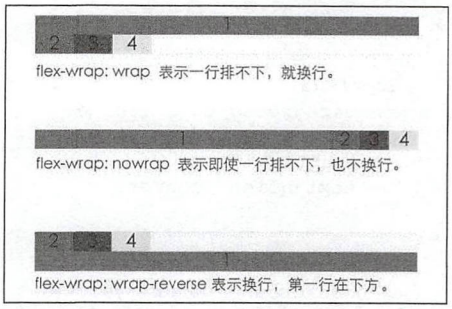


图 3-6 flex-wrap 属性对应的 3 种布局效果

```
<!--pages/viewPage2/viewPage2.wxml-->
<!--flex-wrap 属性:
wrap: 表示一行排不下，就换行。-->
<view class="c_allView1">
  <view class=" c_view1">1</view>
  <view class=" c_view2">2</view>
  <view class=" c_view3">3</view>
  <view class=" c_view4">4</view>
</view>

<!--flex-wrap 属性:
nowrap: 表示即使一行排不下，也不换行。-->
<view class="c_allView2">
  <view class=" c_view1">1</view>
  <view class=" c_view2">2</view>
  <view class=" c_view3">3</view>
  <view class=" c_view4">4</view>
</view>

<!--flex-wrap 属性:
wrap-reverse: 换行，第一行在下方。-->
<view class="c_allView3">
  <view class=" c_view1">1</view>
  <view class=" c_view2">2</view>
  <view class=" c_view3">3</view>
  <view class=" c_view4">4</view>
</view>

/* pages/viewPage2/viewPage2.wxss */
/*flex-wrap 属性 */
.c_view1{
   background-color: red;
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    width: 100%;
    height: 60%;
    text-align: center;
}

.c_view2{
    background-color: green;
    width: 10%;
    height: 40%;
    text-align: center;
}

.c_view3{
    background-color: blue;
    width: 10%;
    height: 60%;
    text-align: center;
}

.c_view4{
    background-color: yellow;
    width: 10%;
    height: 80%;
    text-align: center;
}

/*flex-wrap:wrap: 表示一行排不下，就换行。*/
.c_allView1{
    flex-direction: row;
    flex-wrap: wrap;
    display: flex;
    height: 100%;
}

/*flex-wrap:nowrap: 表示即使一行排不下，也不换行。*/
.c_allView2{
    flex-direction: row;
    flex-wrap: nowrap;
    display: flex;
    height: 100%;
}

/*flex-wrap:wrap-reverse: 换行，第一行在下方。*/
.c_allView3{
    flex-direction: row;
    flex-wrap: wrap-reverse;
    display: flex;
    height: 100%;
}
```

从代码可以看出：

view1 样式设置了红色背景、宽度 100%、高度 60%、文本居中显示。

view2 样式设置了绿色背景、宽度 10%、高度 40%、文本居中显示。

view3 样式设置了蓝色背景、宽度 10%、高度 60%、文本居中显示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

view4 样式设置了黄色背景、宽度 10%、高度 80%、文本居中显示。

3 个容器样式分别为：

c_allView1 样式设置 display: flex 布局模式，flex-direction: row 从左往右排，flex-wrap: wrap 排不下换行处理，高度 100%。

c_allView2 样式设置 display: flex 布局模式，flex-direction: row 从左往右排，flex-wrap: nowrap 排不下也不换行处理，高度 100%。

c_allView3 样式设置 display: flex 布局模式，flex-direction: row 从左往右排，flex-wrap: wrap-reverse 换行处理，第一行在最底下，高度 100%。

4 个视图，在 3 种不同 flex-wrap 属性值的容器中分别呈现出不同的布局效果。

3）justify-content 属性

justify-content 属性定义了项目在主轴上的对齐方式（可以理解为相对于 X 轴的位置）。

justify-content 有 5 个可选的对齐方式：

- flex-start 主轴起点对齐（水平居左对齐，默认对齐方式）。
- flex-end 主轴结束点对齐（水平居右对齐）。
- center 在主轴中居中对齐（水平居中对齐）。
- space-between 两端对齐，两端的子元素贴容器边，其他子元素之间的间隔相等。
- space-around 每个子元素之间的距离相等，两端的子元素距离容器的边距是元素之间间距的 1/2。

justify-content 的对齐方式和主轴的方向有关，下图以 flex-direction 为 row，主轴方式是从左到右，描述 justify-content 属性 5 个值的使用代码如下所示，对应的 5 种布局效果如图 3-7 所示。

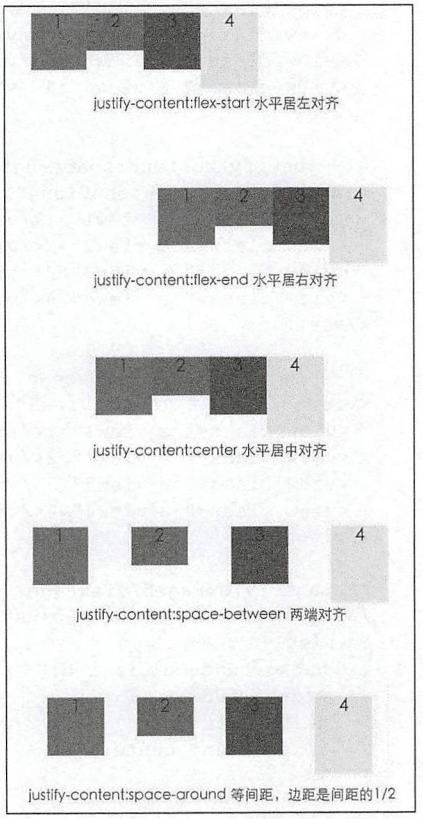


图 3-7 justify-content 属性对应的 5 种布局效果

```
<!--pages/viewPage2/viewPage2.wxml-->
<!-----------------justify-content 属性 ----------------->
<!-- justify-content:flex-start 水平居左对齐 -->
<view class="c_contentView1">
  <view class=" a_view1">1</view>
  <view class=" a_view2">2</view>
  <view class=" a_view3">3</view>
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    <view class=" a_view4">4</view>
  </view>

  <!--justify-content:flex-end 水平居右对齐-->
  <view class="c_contentView2">
    <view class=" a_view1">1</view>
    <view class=" a_view2">2</view>
    <view class=" a_view3">3</view>
    <view class=" a_view4">4</view>
  </view>

  <!--justify-content:center 水平居中对齐-->
  <view class="c_contentView3">
    <view class=" a_view1">1</view>
    <view class=" a_view2">2</view>
    <view class=" a_view3">3</view>
    <view class=" a_view4">4</view>
  </view>

  <!--justify-content:space-between 两端对齐-->
  <view class="c_contentView4">
    <view class=" a_view1">1</view>
    <view class=" a_view2">2</view>
    <view class=" a_view3">3</view>
    <view class=" a_view4">4</view>
  </view>

  <!--justify-content:space-around 等间距，边距是间距的 1/2-->
  <view class="c_contentView5">
    <view class=" a_view1">1</view>
    <view class=" a_view2">2</view>
    <view class=" a_view3">3</view>
    <view class=" a_view4">4</view>
  </view>

  /* pages/viewPage2/viewPage2.wxss *
  /* ---------------------- justify-content 属性  ----------------------  */
  .a_view1{
     background-color: red;
     width: 60px;
     height: 60%;
     text-align: center;
  }

  .a_view2{
     background-color: green;
     width: 60px;
     height: 40%;
     text-align: center;
  }

  .a_view3{
     background-color: blue;
     width: 60px;
     height: 60%;
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    text-align: center;
}

.a_view4{
    background-color: yellow;
    width: 60px;
    height: 80%;
    text-align: center;
}

/*justify-content:flex-start 水平居左对齐*/
.c_contentView1{
    flex-direction: row;
    flex-wrap: nowrap;
    display: flex;
    height: 100px;
    justify-content:flex-start;
}

/*justify-content:flex-end 水平居右对齐*/
.c_contentView2{
    flex-direction: row;
    flex-wrap: nowrap;
    display: flex;
    height: 100px;
    justify-content:flex-end;
}


/*justify-content:center 水平居中对齐*/
.c_contentView3{
    flex-direction: row;
    flex-wrap: nowrap;
    display: flex;
    height: 100px;
    justify-content:center;
}

/*justify-content:space-between 两端对齐*/
.c_contentView4{
    flex-direction: row;
    flex-wrap: nowrap;
    display: flex;
    height: 100px;
    justify-content:space-between;
}

/*justify-content:space-around 等间距，边距是间距的 1/2*/
.c_contentView5{
    flex-direction: row;
    flex-wrap: nowrap;
    display: flex;
    height: 100px;
    justify-content:space-around;
}
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

4）align-items 属性

align-items 定义子元素在侧轴上对齐的方式，有以下几种布局方式，有了主轴和侧轴的方向再加上设置它们的对齐方式，就可以实现大部分的页面布局了。

- stretch 填充整个容器（默认值）。
- flex-start 侧轴的起点对齐。
- flex-end 侧轴的终点对齐。
- center 在侧轴中居中对齐。
- baseline 以子元素的第一行文字对齐。

align-items 设置的对齐方式和侧轴的方向有关，右图以 flex-direction 为 row，侧轴方向是从上到下。描述 align-items 属性的使用代码如下所示，对应的 5 种布局效果如图 3-8 所示。

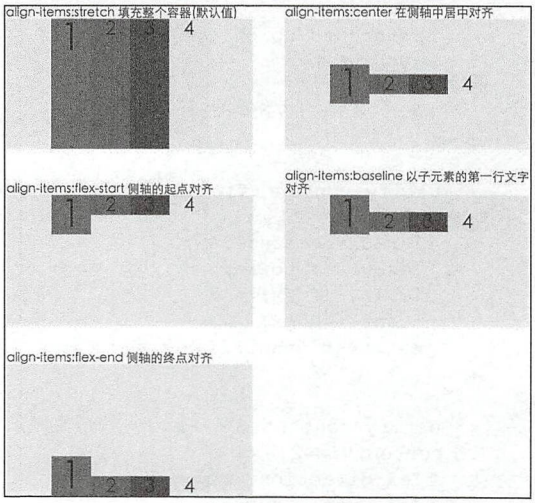


图 3-8 align-items 属性对应的 5 种布局效果

```
    <!--pages/viewPage2/viewPage2.wxml-->
    <!-----------------align-items 属性 ----------------->
    <view>align-items:stretch 填充整个容器(默认值)</view>
    <view class="flex-wrap" style="height: 200px;flex-direction:row;justify-
content: center;align-items:stretch">
      <view class=" align_view1">1</view>
      <view class=" align_view2">2</view>
      <view class=" align_view3">3</view>
      <view class=" align_view4">4</view>
    </view>
    <view>align-items:flex-start 侧轴的起点对齐</view>
    <view class="flex-wrap" style="height: 200px;flex-direction:row;justify-
content: center;align-items:flex-start">
      <view class=" align_view1">1</view>
      <view class=" align_view2">2</view>
      <view class=" align_view3">3</view>
      <view class=" align_view4">4</view>
    </view>
    <view>align-items:flex-end 侧轴的终点对齐</view>
    <view class="flex-wrap" style="height: 200px;flex-direction:row;justify-
content: center;align-items:flex-end">
      <view class=" align_view1">1</view>
      <view class=" align_view2">2</view>
      <view class=" align_view3">3</view>
      <view class=" align_view4">4</view>
    </view>
    <view>align-items:center 在侧轴中居中对齐</view>
    <view class="flex-wrap" style="height: 200px;flex-direction:row;justify-
content: center;align-items:center">
      <view class=" align_view1">1</view>
      <view class=" align_view2">2</view>
```

非卖品，仅供非商业用途或交流学习使用

```
      <view class=" align_view3">3</view>
      <view class=" align_view4">4</view>
    </view>
    <view>align-items:baseline 以子元素的第一行文字对齐</view>
    <view  class="flex-wrap"style="height:  200px;flex-direction:row;justify-
content: center;align-items:baseline">
      <view class=" align_view1">1</view>
      <view class=" align_view2">2</view>
      <view class=" align_view3">3</view>
      <view class=" align_view4">4</view>
    </view>

    /* pages/viewPage2/viewPage2.wxss */
    /* ------------------------ align-items 属性  ------------------------  */
    .align_view1 {
      background-color: red;
      width: 60px;
      text-align: center;
      font-size: 60px;
    }

    .align_view2 {
      background-color: green;
      width: 60px;
      text-align: center;
      font-size: 30px;
    }

    .align_view3 {
      background-color: blue;
      width: 60px;
      text-align: center;
      font-size: 30px;
    }

    .align_view4 {
      background-color: yellow;
      width: 60px;
      text-align: center;
      font-size: 30px;
    }
    /*容器属性*/
    .flex-wrap{
       height: 100px;
       display:flex;
       background-color: #eeeeee;
    }
```

5）align-content 属性

该属性定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。有以下几种布局方式。

- flex-start：与交叉轴的起点对齐（*Y*轴居顶对齐）。
- flex-end：与交叉轴的终点对齐（*Y*轴居底对齐）。

非卖品，仅供非商业用途或交流学习使用

- center：与交叉轴的中点对齐（Y轴居中对齐）。
- space-between：与交叉轴两端对齐，轴线之间的间隔平均分布。
- space-around：每根轴线两侧的间隔都相等，所以轴线之间的间隔比轴线与边框的间隔大一倍。
- stretch（默认值）：轴线占满整个交叉轴，项目被拉伸以适应容器。

align-content 属性使用代码如下所示，布局效果如图 3-9 所示。

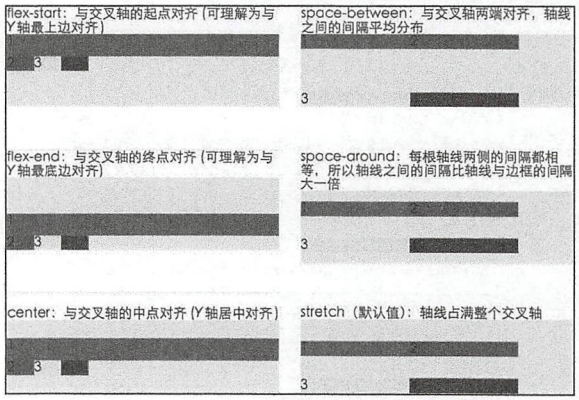


图 3-9 align-content 属性对应的 6 种布局效果

```
<!--pages/viewPage2/viewPage2.wxml-->
<!-----------------第 5 个属性  align-content 属性  ----------------->

<view>flex-start: 与交叉轴的起点对齐（可理解为与 Y 轴最上边对齐）</view>
<view class="a_content_allview" style="align-content: flex-start">
  <view style="background-color: red; width: 100%;height: 30%;">1</view>
    <view style="background-color: green; width: 10%;height: 20%;">2</view>
      <view style="background-color: yellow; width: 10%;height: 20%;">3</view>
      <view style="background-color: blue; width: 10%;height: 20%;">4</view>
</view>

<view>flex-end: 与交叉轴的终点对齐（可理解为与 Y 轴最底边对齐）</view>
<view class="a_content_allview" style="align-content: flex-end">
   <view style="background-color: red; width: 100%;height: 30%;">1</view>
    <view style="background-color: green; width: 10%;height: 20%;">2</view>
      <view style="background-color: yellow; width: 10%;height: 20%;">3</view>
      <view style="background-color: blue; width: 10%;height: 20%;">4</view>
</view>

<view>center: 与交叉轴的中点对齐（Y 轴居中对齐）</view>
<view class="a_content_allview" style="align-content: center">
   <view style="background-color: red; width: 100%;height: 30%;">1</view>
    <view style="background-color: green; width: 10%;height: 20%;">2</view>
      <view style="background-color: yellow; width: 10%;height: 20%;">3</view>
      <view style="background-color: blue; width: 10%;height: 20%;">4</view>
</view>

<view>space-between: 与交叉轴两端对齐，轴线之间的间隔平均分布</view>
<view class="a_content_allview" style="align-content: space-between">
   <view style="background-color: red; width: 40%;height: 20%;">1</view>
    <view style="background-color: green; width: 40%;height: 20%;">2</view>
      <view style="background-color: yellow; width: 40%;height: 20%;">3</view>
      <view style="background-color: blue; width: 40%;height: 20%;">4</view>
</view>

<view>space-around: 每根轴线两侧的间隔都相等，所以轴线之间的间隔比轴线与边框的间隔
```

非卖品，仅供非商业用途或交流学习使用

```
大一倍</view>
    <view class="a_content_allview" style="align-content: space-around">
       <view style="background-color: red; width: 40%;height: 20%;">1</view>
        <view style="background-color: green; width: 40%;height: 20%;">2</view>
          <view style="background-color: yellow; width:40%;height: 20%;">3</view>
          <view style="background-color: blue; width: 40%;height: 20%;">4</view>
    </view>

    <view>stretch(默认值): 轴线占满整个交叉轴</view>
    <view class="a_content_allview" style="align-content: stretch">
       <view style="background-color: red; width: 40%;height: 20%;">1</view>
        <view style="background-color: green; width: 40%;height: 20%;">2</view>
          <view style="background-color: yellow; width: 40%;height: 20%;">3</view>
          <view style="background-color: blue; width: 40%;height: 20%;">4</view>
    </view>

    /* ---------------------- 第 5 个属性 align-content 属性 ----------------- */
    /* pages/viewPage2/viewPage2.wxss */
    .a_content_allview{
        background-color: #eeeeee;
        /*设置弹性布局的方向*/
        flex-direction: row;
        /*是否换行*/
        flex-wrap: wrap;
        /*设置子元素排列*/
        justify-content: flex-start;
        /*设置按照弹性布局展示*/
        display: flex;
        /*设置子元素交叉轴排列*/
        /*在 wxml 组件那设置 align-content: stretch;*/
        align-items: flex-end;
        /*设置高度为 100px*/
        height: 100px;
    }
```

至此，flex 容器的属性就介绍完了，总结如下：

（1）flex-direction 设置容器内子元素即 flex-item 的排列方向。

（2）flex-wrap 设置容器内子元素是否换行。

（3）justify-content 设置子元素在横轴（即 *X* 轴）的排列位置。

（4）align-items 设置子元素在 *Y* 轴的排列位置。

（5）align-content 设置子元素在多个主轴线上排列的位置。

在 flex 容器中，每个子元素就是一个 flex-item。接下来介绍 flex 容器的子元素 flex-item 的属性。

flex-item（项目）的属性

1）order 属性

order 属性定义项目的排列顺序。数值越小，排列越靠前。如果 item 没有设置 order 属性，则 order 默认为 0，例如给容器添加 4 个组件 view1、view2、view3、view4，它们的 order 分别是 4、3、2、1，显示出来之后，会看到 view4 排在第一个位置。具体代码如下所

非卖品，仅供非商业用途或交流学习使用

示，显示效果如图 3-10 所示。

```
<!--pages/viewPage3/viewPage3.wxml-->

<!--  order 属性  -->
<view class="view-content">
  <view class="c_view" style="background-color: red;order:4">1</view>
  <view class="c_view" style="background-color: green;order:3">2</view>
  <view class="c_view" style="background-color: blue;order:2">3</view>
  <view class="c_view" style="background-color: yellow;order:1">4</view>
</view>

.view-content {
  display: flex;
  flex-direction: row;
  align-items: center;
  justify-content: space-between;
}

.c_view {
  width: 20%;
  height: 60px;
  text-align: center;
}
```

2）flex-grow 属性

flex-grow 属性定义项目的放大比例，是指存在剩余空间时，项目将按照 flex-grow 数值放大分取剩余空间。如果一个项目的 flex-grow 属性为 2，其他项目都为 1，则前者占据的剩余空间将比其他项多一倍。flex-grow 属性默认为 0，即使存在剩余空间，也不放大。如果所有项目的 flex-grow 属性都为 1，则等分剩余的空间。使用代码如下所示，显示效果如图 3-11 所示。

```
<!--pages/viewPage3/viewPage3.wxml-->
<!--  flex-grow 属性  -->
<view class="view-content">
  <view style="background-color: red;flex-grow:4">1</view>
  <view style="background-color: green;flex-grow:2">2</view>
  <view style="background-color: blue;flex-grow:2">3</view>
  <view  style="background-color: yellow;flex-grow:1">4</view>
</view>

/* pages/viewPage3/viewPage3.wxss */
.view-content {
  display: flex;
  flex-direction: row;
  align-items: center;
  justify-content: space-between;
}
```

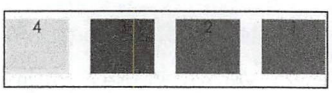


图 3-10　order 属性显示效果

图 3-11　flex-grow 属性显示效果

3）flex-shrink 属性

flex-shrink 属性定义了项目的缩小比例，即如果空间不足，该项目将缩小。flex-shrink 属性默认为 1，表示所有项目在空间不足时，都将等比例缩小。如果一个项目的 flex-shrink 属性为 0，其他项目都为 1，则空间不足时，前者不缩小。使用代码如下所示，显示效果如图 3-12 所示。

图 3-12 flex-shrink 属性显示效果

```
<!--pages/viewPage3/viewPage3.wxml-->
<!--  flex-shrink 属性  -->
<view class="view-content">
  <view class="c_view2" style="background-color: red;flex-shrink:3">1</view>
  <view class="c_view2" style="background-color: green;flex-shrink:3">2</view>
  <view class="c_view2" style="background-color: blue;flex-shrink:2">3</view>
  <view class="c_view2" style="background-color: yellow;flex-shrink:2">4</view>
</view>

/* pages/viewPage3/viewPage3.wxss */
.c_view2 {
  width: 100%;
  height: 60px;
  text-align: center;
}
```

4）flex-basis 属性

flex-basis 属性定义了在分配多余空间之前，项目占据的主轴空间（main size）。如果主轴有多余空间，项目按照 flex-basis 值进行放大占有，它的默认值为 0，即项目的本来大小。使用代码如下所示，显示效果如图 3-13 所示。

```
<!--pages/viewPage3/viewPage3.wxml-->
<!--  flex-basis 属性  -->
<view>flex-basis 属性</view>
<view class="view-content1">
  <view class="a_view1" style="background-color: red;flex-basis:100%">1</view>
  <view class="a_view2" style="background-color: green;flex-basis:50%">2</view>
  <view class="a_view3" style="background-color: blue;flex-basis:20%">3</view>
  <view class="a_view4" style="background-color: yellow;flex-basis:10%">4</view>
</view>

/* pages/viewPage3/viewPage3.wxss */
/*    flex-basis 属性    */
.view-content1 {
  display: flex;
  flex-direction: row;
  width: 100%;
  height: 200px;
  background-color: #eeeeee;
}
.a_view1 {
  width: 100%;
  height: 60px;
  text-align: center;
```

```
}
.a_view2 {
  width: 100%;
  height: 80px;
  text-align: center;
}
.a_view3 {
  width: 100%;
  height: 100px;
  text-align: center;
}
.a_view4 {
  width: 100%;
  height: 120px;
  text-align: center;
}
```

5）align-self 属性

align-self 属性允许单个项目有与其他项目不一样的对齐方式，可覆盖 align-items 属性。默认值为 auto，表示继承父元素的 align-items 属性，如果没有父元素，则等同于 stretch。代码如下所示，显示效果如图 3-14 所示。

```
<!--pages/viewPage3/viewPage3.wxml-->
<!--  align-self 属性  -->
<view>align-self 属性</view>
<view class="view-content1">
  <view class="a_view1" style="background-color: red;flex-basis:100%;align-
self:center"> 1</view>
  <view class="a_view2" style="background-color: green;flex-basis:50%">2</view>
  <view class="a_view3" style="background-color: blue;flex-basis:20%">3</view>
  <view class="a_view4" style="background-color: yellow;flex-basis:10%">4</view>
</view>
```

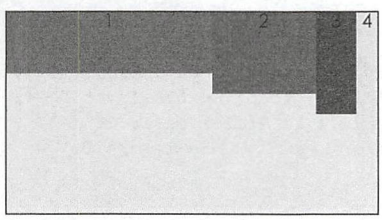


图 3-13　flex-basis 属性显示效果

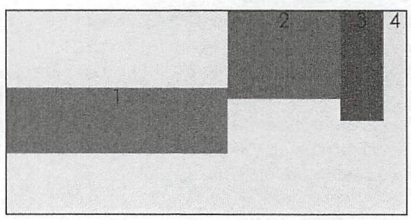


图 3-14　align-self 属性显示效果

6）flex 权重属性

可选项有 1、2、3、4、5 等，必须与 display 属性一起结合使用，单独使用 flex 无效。具体代码如下所示，显示效果如图 3-15 所示。

```
<!--pages/viewPage3/viewPage3.wxml-->
<!--  flex 权重属性  -->
<view>flex 权重属性</view>
<view class="view-content3">
```

```
      <image   class="flex3"   style="width:100px;height:100px"   src="http://
wx.leadingdo.com/images/book1.png"> </image>
      <image   class="flex2"   style="width:100px;height:100px"   src="http://
wx.leadingdo.com/images/book2.png"> </image>
    </view>

    /* pages/viewPage3/viewPage3.wxss */
    /*权重布局*/
    .view-content3 {
      display: flex;
    }

    .flex5 {
      flex: 5;
    }

    .flex4 {
      flex: 4;
    }

    .flex3 {
      flex: 8;
    }

    .flex2 {
      flex: 2;
    }

    .flex1 {
      flex: 1;
    }
```

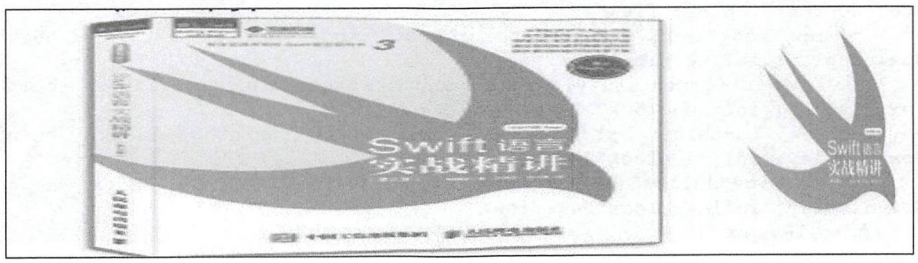


图 3-15 flex 权重属性显示效果

3.2.2 scroll-view

scroll-view 是容器组件，如果内部组件超过了 scroll-view 的高度或宽度，可以通过水平、垂直滚动来显示内部组件。这样，scroll-view 可以用来展示大量的内容，并且可以通过滚动查看所有的内容。在使用的时候，开发者可以设置 scroll-view 的高度或宽度，并设置是否允许水平滚动或者垂直滚动。

scroll-view 主要属性如表 3-5 所示。

表 3-5　　scroll-view 的主要属性

属性名	类型	默认值	说明
scroll-x	Boolean	false	允许水平滚动
scroll-y	Boolean	false	允许垂直滚动
upper-threshold	Number	50	距顶部/左边多远时（单位为 px），触发 scrolltoupper 事件
lower-threshold	Number	50	距底部/右边多远时（单位为 px），触发 scrolltolower 事件
scroll-top	Number		设置垂直滚动条位置
scroll-left	Number		设置水平滚动条位置
scroll-into-view	String		值应为某子元素 id，滚动到该元素，元素顶部对齐滚动区域顶部
bindscrolltoupper	EventHandle		滚动到顶部/左边，会触发 scrolltoupper 事件
bindscrolltolower	EventHandle		滚动到底部/右边，会触发 scrolltolower 事件
bindscroll	EventHandle		滚动时触发，event.detail = {scrollLeft, scrollTop, scrollHeight, scrollWidth, deltaX, deltaY}

1）scroll-x

设置 scroll-x ="true"，scroll-view 才可以实现水平滚动，默认 scroll-x ="false"。具体代码如下所示，显示效果如图 3-16 所示。

```
    <!--pages/viewPage4/viewPage4.wxml-->
    <view>水平滚动</view>
    <scroll-view scroll-x ="true" style="width: 100%;height: 100px; white-
space: nowrap;display: flex">
        <view id="red" style="background: red; width: 200px; height:
100px;display: inline-block">1</view>
        <view id="green" style="background: green; width: 200px; height:
100px;display: inline-block">2</view>
        <view id="blue" style="background: blue; width: 200px; height:
100px;display: inline-block">3</view>
        <view id="yellow" style="background: yellow; width: 200px; height:
100px;display: inline-block">4</view>
    </scroll-view>
```

2）scroll-y

设置 scroll-y ="true"，scroll-view 才可以实现垂直滚动，默认 scroll-y ="false"。具体代码如下所示，显示效果如图 3-17 所示。

```
    <!--pages/viewPage4/viewPage4.wxml-->
    <view>垂直滚动，这里必须设置高度</view>
    <scroll-view scroll-y="true" style="width: 100%;height: 400px">
      <view id="red" style="background: red; width: 100%; height: 400px"></view>
      <view id="green" style="background: green; width: 100%; height: 400px"></view>
      <view id="blue" style="background: blue; width: 100%; height: 400px">
</view>
      <view id="yellow" style="background:yellow; width: 100%; height: 400px">
</view>
    </scroll-view>
```

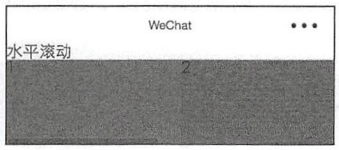


图 3-16 scroll-view 水平滚动

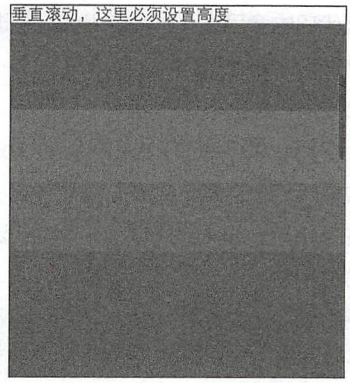


图 3-17 scroll-view 垂直滚动

关于 display 属性的几点说明：

（1）display:none; // 不显示。

（2）display:block; // 此元素将显示为块级元素，此元素前后会带有换行符。

（3）display:inline; // 默认。此元素会被显示为内联元素，元素前后没有换行符。

（4）display:inline-block; // 行内块元素。

3）upper-threshold 和 bindscrolltoupper

upper-threshold 属性设置一个数值，单位是 px，bindscrolltoupper 用来绑定一个事件，这两者基本是结合使用。使用代码如下：

```
    <scroll-view scroll-y="true" style="width: 100%;height: 400px" upper-
threshold="20" bindscrolltoupper="bindscrolltoupper" >

    // pages/viewPage4/viewPage4.js
      //滚动到顶部/左边触发
      bindscrolltoupper:function(){
        console.log("------->滚动到顶部/左边");
      },
    })
```

上面代码的意思是：scroll-view 滚动到距顶部/左边的距离等于 20 像素的时候触发执行 bindscrolltoupper 绑定的事件。

4）lower-threshold 和 bindscrolltolower

lower-threshold 属性设置一个数值，单位是 px，bindscrolltolower 用来绑定一个事件，这两者基本是结合使用。使用代码如下：

```
    <scroll-view scroll-y="true" style="width: 100%;height: 400px" lower-
threshold="50" bindscrolltolower="bindscrolltolower">

    // pages/viewPage4/viewPage4.js
      //滚动到底部/右边触发
      bindscrolltolower:function(){
        console.log("------->滚动到底部/右边");
      }
    })
```

上面代码的意思是：scroll-view 滚动到距底部/右边的距离等于 50 像素的时候触发执行 bindscrolltolower 绑定的事件。

5）scroll-top

scroll-top 对于垂直方向来说，scroll-view 滚动条的初始位置为 0，也就是在最顶端，如果要改变滚动条的默认位置，需要设置 scroll-top 属性。该属性默认的属性值为 0，也就是滚动条在最顶端。如果该属性值不为 0，滚动条会向下滚动。下面代码设置了 scroll-top 的属性值是 60。

```
    <scroll-view scroll-y="true" style="width: 100%;height: 400px" upper-threshold="20" bindscrolltoupper="bindscrolltoupper" lower-threshold="50" bindscrolltolower="bindscrolltolower" scroll-top="60">
```

从 scroll-view 滚动条的位置可以看出，设置 scroll-top="60"后，滚动条位置没有在最顶部，而是距离顶部有 60 像素的位置，如图 3-18 所示。

6）scroll-left

scroll-left 对于水平方向来说，scroll-view 滚动条的初始位置为 0，也就是在最左边，如果要改变滚动条的默认位置，需要设置 scroll-left 属性。该属性默认的属性值为 0，也就是滚动条在最左边。如果该属性值不为 0，滚动条会向右滚动。下面代码设置了 scroll-left 的属性值是 60。

```
    <scroll-view scroll-x="true" style="width: 100%;height: 100px; white-space: nowrap;display: flex" scroll-left="60">
```

从 scroll-view 滚动条的位置可以看出，设置 scroll-left ="60"后，滚动条位置没有在最左边，而是距离左边有 60 像素的位置，如图 3-19 所示。

7）scroll-into-view

如果想让 scroll-view 一开始就滚动到某一个子视图，需要使用 scroll-into-view 属性，该属性需要指定一个子视图的 id。例如，下面的布局代码设置了 scroll-into-view 的属性值为 blue。

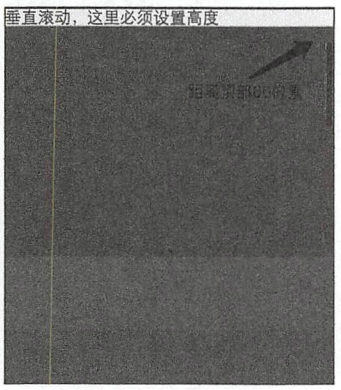


图 3-18 scroll-top="60"效果

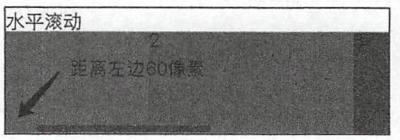


图 3-19 scroll-left="60"效果

```
    <scroll-view scroll-y="true" style="width: 100%;height: 400px" upper-
threshold="20" bindscrolltoupper="bindscrolltoupper"lower-threshold="50" bind-
scrolltolower="bindscrolltolower" scroll-top="60" scroll-into-view="blue">
```

当系统加载 scroll-view 组件时，会直接滚动到第 3 个子组件 blue。如果同时设置了 scroll-top 和 scroll-into-view 属性，scroll-into-view 的优先级更高。

8）bindscroll

bindscroll 用于绑定一个事件，只要 scroll-view 滚动，都会触发该事件。代码如下所示：

```
    <scroll-view scroll-y="true" style="width: 100%;height: 400px" upper-
threshold="20" bindscrolltoupper="bindscrolltoupper" lower-threshold="50"
bindscrolltolower= "bindscrolltolower" scroll-top="60" scroll-into-
view="blue" bindscroll ="bindscroll"
    >

    // pages/viewPage4/viewPage4.js
    Page({
      data:{},
      //滚动事件
      bindscroll:function(event){
        console.log("------->",event);
      }
    })
```

触发事件时会带 event 对象，可以从 event.detail 中获取 scroll-view 的 scrollLeft、scrollTop、scrollHeight、scrollWidth、deltaX、deltaY 值，如图 3-20 所示。

上面所讲到的 upper-threshold、lower-threshold、scroll-top、scroll-left、scroll-into-view 在代码中设置的参数都是固定值，这里我们可以结合第二章的数据绑定，动态设置它们的参数，实现代码控制各个属性的参数。这里以 scroll-left 为例，设置 scroll-left ="{{scrollLeft}}"，scrollLeft 从数据中获取。为了更好地展示，添加一个按钮，绑定一个单击事件，代码如下所示：

```
    <button bindtap="addTap"> scroll-left 加 20 像素 </button>
```

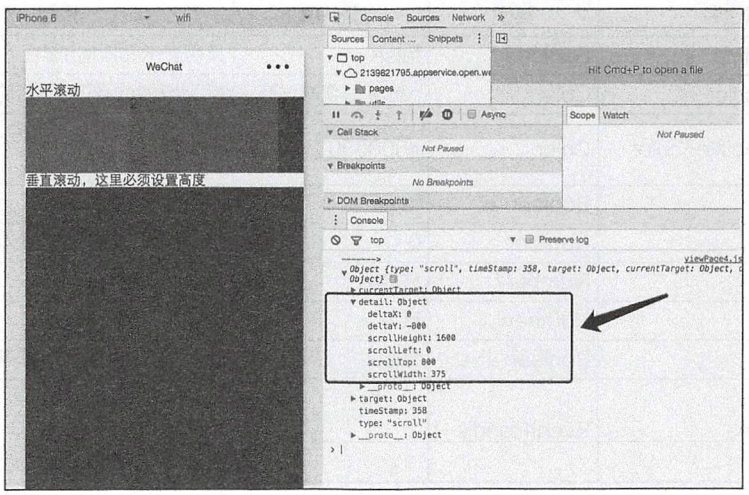


图 3-20 event 对象属性

addTap 事件，每次单击 scrollLeft 会增加 20 像素。addTap 事件代码如下所示：

```
//scroll-left 加 20 像素
  addTap:function(event){
    this.setData({
        scrollLeft: this.data.scrollLeft+20
      });
    console.log("this.data.scrollTop:" + this.data.scrollTop);
  }
```

官方文档指出在 scroll-view 中不能使用 textarea、map、canvas、video 组件。在滚动 scroll-view 时会阻止页面回弹，所以在 scroll-view 中滚动无法触发 onPullDown Refresh（页面的下拉刷新事件）。若要使用下拉刷新，请使用页面的滚动，而不是 scroll-view，这样也能通过单击顶部状态栏回到页面顶部。

如果想让 scroll-view 隐藏滚动条，可以在“.wxss”文件中添加 CSS 样式，代码如下所示：

```
::-webkit-scrollbar {
  width: 0;
  height: 0;
  color: transparent;
}
```

3.2.3 swiper 与 swiper-item

swiper 是微信小程序封装的幻灯片轮播功能，又称滑动面板。开发者只需通过简单地配置参数，就能完成滑动的需求，其中基本的属性定义如表 3-6 所示。

表 3-6　　swiper 的属性

属性名	类型	默认值	说明
indicator-dots	Boolean	false	是否显示面板指示点
indicator-color	Color	rgba(0, 0, 0, .3)	指示点颜色（这个属性目前暂未启用）
indicator-active-color	Color	#000000	当前选中的指示点颜色（这个属性目前暂未启用）
autoplay	Boolean	false	是否自动切换
current	Number	0	当前所在页面的 index
interval	Number	5000	自动切换时间间隔
duration	Number	500	滑动动画时长
circular	Boolean	false	是否采用衔接滑动，循环滑动
bindchange	EventHandle		current 改变时会触发 change 事件，event.detail = {current: current, source: source}

从公共库 v1.4.0 开始，change 事件返回 detail 中包含一个 source 字段，表示导致变更

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

的原因，可能值如下：

- autoplay 自动播放导致 swiper 变化。
- touch 用户滑动引起 swiper 变化。
- 其他原因将用空字符串表示。

swiper 中只能放置<swiper-item>组件，否则会导致未定义的行为。放置其他组件会被自动删除，swiper 组件<swiper-item>自动设置宽高为 100%。

swiper 的使用示例代码如下：

```
<!--pages/viewPage5/viewPage5.wxml-->
<swiper indicator-dots="{{indicatorDots}}" autoplay="{{autoplay}}" interval="{{interval}}" duration="{{duration}}" circular="{{circular}}" bindchange="bindchange">
  <block wx:for="{{imgUrls}}">
    <swiper-item>
      <image src="{{item}}" class/>
    </swiper-item>
  </block>
</swiper>

/* pages/viewPage5/viewPage5.wxss */
swiper{
   height: 150px;
   width:100%;
}

// pages/viewPage5/viewPage5.js
Page({
  data: {
    imgUrls: [
      'http://wx.leadingdo.com/images/image1.jpg',
      'http://wx.leadingdo.com/images/image2.jpg',
      'http://wx.leadingdo.com/images/image3.jpg'
    ],
    indicatorDots: true,//是否显示面板指示点
    autoplay: true,//是否自动切换
    interval: 3000,//自动切换时间间隔
    duration: 1000,//滑动动画时长
    circular: true //是否采用衔接滑动,循环滑动
  },

  bindchange:function(e){
     console.log(e);
  }
})
```

代码使用了动态数据绑定，.js 文件 data 数据设置了 swiper 属性。

```
indicatorDots: true,//是否显示面板指示点
autoplay: true,//是否自动切换
interval: 3000,//自动切换时间间隔
duration: 1000,//滑动动画时长
circular: true //是否采用衔接滑动,循环滑动
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

另外，swiper 设置了 bindchange="bindchange"，绑定了滑动改变事件。在每次滑动切换页面的时候，会触发 bindchange 事件，事件返回 e 对象。开发者可以通过 e.detail 获得 swiper 当前页面 index 值的 current 属性，如图 3-21 所示。

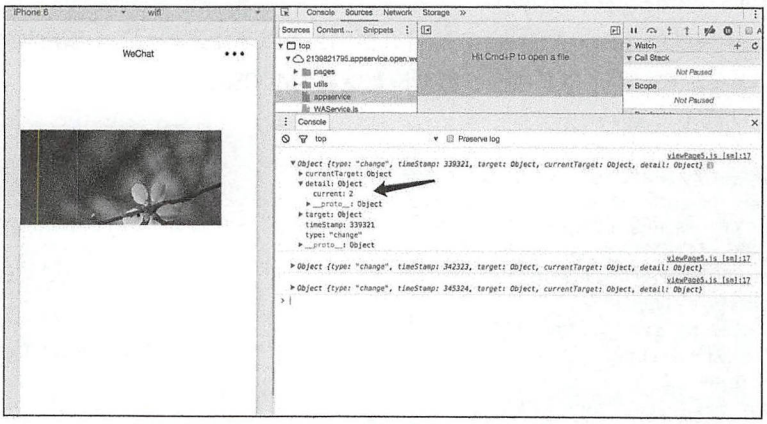


图 3-21　bindchange 事件输出对象的属性

开发者可以自定义 swiper 的底部指示点，代码如下所示，效果图如图 3-22 所示。

```
<!--pages/viewPage5/viewPage5.wxml-->
<!--自定义 swiper 的底部指示点-->
<view>
<swiper indicator-dots="true"
  autoplay="true" interval="3000" duration="1000">
  <block wx:for="{{imgUrls}}">
    <swiper-item>
      <image mode="aspectFill" src="{{item}}" class="slide-image"/>
    </swiper-item>
  </block>
</swiper>
</view>

/* pages/viewPage5/viewPage5.wxss */
/*自定义 swiper 的底部指示点*/
/*当前页面属性*/
page {
  display: block;
  min-height: 100%;
    background: #eeeeee;
}

/*swiper 属性*/
swiper{
 height:480rpx;
}

/*slide 图片属性*/
.slide-image{
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    width: 100%;
    height:480rpx;
}


/*开启底部滚动条*/
page .wx-swiper-dots.wx-swiper-dots-horizontal{
    margin-bottom: -4rpx;
}

/*滚动条属性*/
page .wx-swiper-dot{
    width: 252rpx;
    display: inline-flex;
    height: 18rpx;
    vertical-align: sub;

}

/*默认滚动条属性*/
page .wx-swiper-dot::before{
    content: '';
    background: #e5e5e5;
    flex-grow: 1;
}

/*选择时的滚动条属性*/
page .wx-swiper-dot-active::before{
    content: '';
    background:rgba(240,0,0,0.8);
    flex-grow: 1;
}

// pages/viewPage5/viewPage5.js
Page({
  data: {
    imgUrls: [
      'http://wx.leadingdo.com/images/image1.jpg',
      'http://wx.leadingdo.com/images/image2.jpg',
      'http://wx.leadingdo.com/images/image3.jpg'
    ],
    indicatorDots: true,//是否显示面板指示点
    autoplay: true,//自动切换
    interval: 3000,//自动切换时间间隔
    duration: 1000,//滑动动画时长
    circular: true //是否采用衔接滑动,循环滑动
  }
})
```

项目中一般使用 swiper 来做轮播图，除了实现广告效果之外，另一种常用的功能就是利用 swiper 定制 tab 切换效果。

具体代码如下所示，展示效果如图 3-23 所示，单击按钮会切换 swiper，滑动 swiper 也会对应切换按钮。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

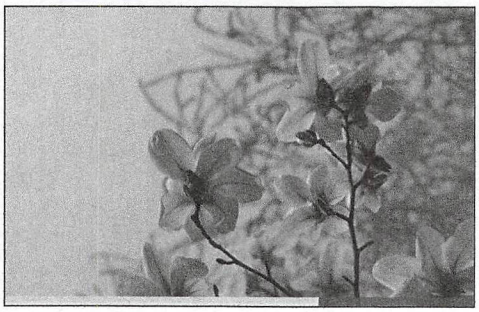


图 3-22 自定义 swiper 的底部指示点

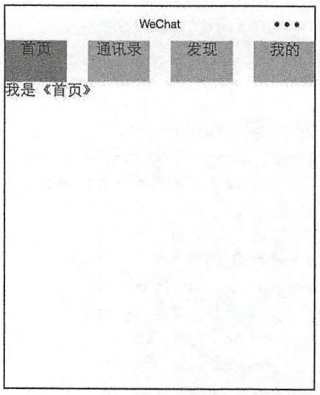


图 3-23 swiper 定制 tab 切换效果

```
    <!--pages/viewPage6/viewPage6.wxml-->
    <view class="flex-row" style="display: flex;">
      <view class="{{currentTab==0 ? 'flex-view-item1' : 'flex-view-item'}}"
bindtap="tap0">首页</view>
      <view class="{{currentTab==1 ? 'flex-view-item1' : 'flex-view-item'}}"
bindtap="tap1">通讯录</view>
      <view class="{{currentTab==2 ? 'flex-view-item1' : 'flex-view-item'}}"
bindtap="tap2">发现</view>
      <view class="{{currentTab==3 ? 'flex-view-item1' : 'flex-view-item'}}"
bindtap="tap3">我的</view>
    </view>
    <swiper current="{{currentTab}}" class="swiper-box" duration="300" style=
"height:{{winHeight - 31}}px" bindchange="bindChange">

      <!-- 我是哈哈 -->
      <swiper-item>
        <view>我是《首页》</view>
      </swiper-item>

      <!-- 我是呵呵 -->
      <swiper-item>
        <view>我是《通讯录》</view>
      </swiper-item>

      <!-- 我是嘿嘿 -->
      <swiper-item>
        <view>我是《发现》</view>
      </swiper-item>

      <!-- 我是哼哼-->
      <swiper-item>
        <view>我是《我的》</view>
      </swiper-item>
    </swiper>

    /* pages/viewPage6/viewPage6.wxss */

    /*按钮所在容器的属性设置*/
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
.flex-row {
  display: flex;
  flex-direction: row;
  align-items: center;
  justify-content: space-between;
}

/*按钮默认状态下属性*/
.flex-view-item {
  width: 20%;
  height: 50px;
  background-color: #ffa589;
  text-align: center;
}
/*按钮选中状态下属性*/
.flex-view-item1 {
  width: 20%;
  height: 50px;
  background-color: #ff5400;
  text-align: center;
}

// pages/viewPage6/viewPage6.js
Page({
  data:{
    currentTab : 0  //初始化 currentTab=0
  },
  //第 1 个按钮单击事件，修改 currentTab 值
  tap0:function(event){
    this.setData({
        currentTab: 0
    });
  },
  // 第 2 个按钮单击事件，修改 currentTab 值
  tap1:function(event){
    this.setData({
        currentTab: 1
    });
  },
  // 第 3 个按钮单击事件，修改 currentTab 值
  tap2:function(event){
    this.setData({
        currentTab: 2
    });
  },
  // 第 4 个按钮单击事件，修改 currentTab 值
  tap3:function(event){
    this.setData({
        currentTab: 3
    });
  },
  // swiper 滑动事件，修改 currentTab 值，实现切换 tab
  bindChange: function( e ) {
    var that = this;
    that.setData( { currentTab: e.detail.current });
  },
})
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

3.2.4 movable-area与movable-view

movable-area 是 movable-view 的可移动区域，该组件从基础库 1.2.0 开始支持。movable-area 必须设置 width 和 height 属性，默认值为 10px。内容可以设置 movable-view 组件，在 movable-area 内部可以拖曳滑动。

movable-view 可移动视图容器也必须设置 width 和 height 属性，默认值为 10px。属性如表 3-7 所示。

表 3-7 movable-view 属性

属性名	类型	默认值	说明
direction	String	none	movable-view 的移动方向，属性值有 all、vertical、horizontal、none
inertia	Boolean	false	movable-view 是否带有惯性
out-of-bounds	Boolean	false	超过可移动区域后，movable-view 是否还可以移动
x	Number		定义 X 轴方向的偏移，如果 x 的值不在可移动范围内，会自动移动到可移动范围；改变 x 的值会触发动画
y	Number		定义 Y 轴方向的偏移，如果 y 的值不在可移动范围内，会自动移动到可移动范围；改变 y 的值会触发动画
damping	Number	20	阻尼系数，用于控制 x 或 y 改变时的动画和过界回弹的动画，值越大移动越快
friction	Number	2	摩擦系数，用于控制惯性滑动的动画，值越大摩擦力越大，滑动越快停止；必须大于 0，否则会被设置成默认值

movable-view 默认为绝对定位，top 和 left 属性为 0px。movable-view 必须在<movable-area/>组件中，并且必须是直接子节点，否则不能移动。当 movable-view 小于 movable-area 时，movable-view 的移动范围在 movable-area 内；当 movable-view 大于 movable-area 时，movable-view 的移动范围必须包含 movable-area（X 轴方向和 Y 轴方向分开考虑）。

具体代码如下所示，代码运行效果，可以参考源代码实例 Ch3.2。

```
// pages/viewPage7/viewPage7.js
Page({
 /**
  * 页面的初始数据
  */
 data: {
  x: 0,
  y: 0
 },

 //单击事件
 tap: function (e) {
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        this.setData({
          x: 30,
          y: 30
        });
      },
    })


    <!--pages/viewPage7/viewPage7.wxml-->
    <view class="section">
      <view class="section">movable-view 区域小于 movable-area</view>
      <movable-area style="height: 200px;width: 200px;background: red;">
        <movable-view style="height: 50px; width: 50px; background: blue;"
x="{{x}}" y="{{y}}" direction="all">
        </movable-view>
      </movable-area>
      <view class="section">
        <button size="mini" bindtap="tap">click me to move to (30px, 30px)</button>
      </view>
      <view class="section">movable-view 区域大于 movable-area</view>
      <movable-area style="height: 200px;width: 200px;background: red;"
direction="all">
        <movable-view style="height: 50px; width: 300px; background: blue;"
x="{{x}}" y="{{y}}" direction="all">
        </movable-view>
      </movable-area>
    </view>

    /* pages/viewPage7/viewPage7.wxss */
    .section {
      margin-top: 20px;
    }
```

3.2.5 cover-view 与 cover-image

cover-view 组件从基础库 1.4.0 开始支持，是覆盖在原生组件之上的文本视图，可覆盖的原生组件包括 map、video、canvas，支持嵌套。

cover-view 组件内可以添加 cover-image 组件，cover-image 组件是覆盖在原生组件之上的图片视图，可覆盖的原生组件同 cover-view，并支持嵌套在 cover-view 里。

cover-image 组件的属性，如表 3-8 所示。

表 3-8 cover-image 属性

属性名	类型	默认值	说明
src	String		图标路径，支持临时路径。暂不支持 base64 与网络地址

cover-image 组件一些注意事项如下：

1. 只可嵌套在原生组件 map、video、canvas 内，需避免嵌套在其他组件内。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

2. 事件模型遵循冒泡模型，但不会冒泡到原生组件。

3. 文本建议都套上 cover-view 标签，避免排版错误。

4. 只支持基本的定位、布局、文本样式。不支持设置单边的 border、opacity、background-image 等。

5. 建议子节点不要溢出父节点。

6. 暂不支持 CSS 动画。

具体代码如下所示，显示效果参考源代码 Ch3.2。

```
Page({

  /**
   * 生命周期函数——监听页面初次渲染完成
   */
  onReady() {
    this.videoCtx = wx.createVideoContext('myVideo')
  },

  //开始
  play() {
    this.videoCtx.play()
  },

  //暂停
  pause() {
    this.videoCtx.pause()
  }
})

<!--pages/viewPage8/viewPage8.wxml-->

<!-- 视频组件 -->
<video id="myVideo" src="http://wxsnsdy.tc.qq.com/105/20210/snsdyvideodownload?
filekey=30280201010421301f0201690402534804102ca905ce620b1241b726bc41dcff44e0
0204012882540400&bizid=1023&hy=SH&fileparam=302c020101042530230204136ffd9302
0457e3c4ff02024ef202031e8d7f02030f42400204045a320a0201000400"
controls="{{false}}" event- model="bubble">

  <!-- 在视频组件上面添加播放按钮、暂停按钮、时间 -->
  <cover-view class="controls">
    <cover-view class="play" bindtap="play">
      <cover-image class="img" src="../images/play.png" />
    </cover-view>
    <cover-view class="pause" bindtap="pause">
      <cover-image class="img" src="../images/pause.jpg" />
    </cover-view>
    <cover-view class="time">00:00</cover-view>
  </cover-view>
</video>

/* pages/viewPage8/viewPage8.wxss */
.controls {
  position: relative;
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    top: 50%;
    height: 50px;
    margin-top: -25px;
    display: flex;
  }
  .play,.pause,.time {
    flex: 1;
    height: 100%;
  }
  .time {
    text-align: center;
    background-color: rgba(0, 0, 0, .5);
    color: white;
    line-height: 50px;
  }
  .img {
    width: 40px;
    height: 40px;
    margin: 5px auto;
  }
```

3.3 基础内容

基础内容包括 icon（图标）、text（文本）、rich-text（富文本）和 progress（进度条）。

3.3.1 icon

icon 可以直接用微信组件默认的图标，默认是 iconfont 格式的，从 WeUI 那边沿袭过来的一种做法。开发者根据提供的、所支持的图标样式，按照需求在此基础上去修改大小和颜色。

icon 有 3 个属性，如表 3-9 所示。

表 3-9 icon 属性

属性名	类型	默认值	说明
type	String		icon 的类型，有效值：success、success_no_circle、info、warn、waiting、cancel、download、search、clear
size	Number	23	icon 的大小，单位为 px
color	Color		icon 的颜色，同 CSS 的 color

icon 的类型一共有 success、info、warn、waiting、safe_success、safe_warn、success_circle、success_no_circle、waiting_circle、circle、download、info_circle、cancel、search、clear 等十几种形式。

icon 的使用很简单，就是定义 type 和设置 size、color 属性即可，代码如后几页所示，显示效果如图 3-24 所示。

非卖品，仅供非商业用途或交流学习使用

图 3-24 icon 各 type 效果

```
    <!--pages/iconPage/iconPage.wxml-->
    <view>成功图标
      <block wx:for="{{iconSizeColor}}">
        <icon  class="flex-row"  type="success"  size="{{item.size}}"  color=
"{{item.color}}" />
      </block>
    </view>

    <view>提示信息图标
      <block wx:for="{{iconSizeColor}}">
        <icon  class="flex-row"  type="info"  size="{{item.size}}"  color=
"{{item.color}}" />
      </block>
    </view>

    <view>警告图标
      <block wx:for="{{iconSizeColor}}">
        <icon  class="flex-row"  type="warn"  size="{{item.size}}"  color=
"{{item.color}}" />
      </block>
    </view>
```

非卖品，仅供非商业用途或交流学习使用

```
    <view>等待图标
      <block wx:for="{{iconSizeColor}}">
        <icon  class="flex-row"  type="waiting"  size="{{item.size}}"  color=
"{{item.color}}" />
      </block>
    </view>

    <view>安全成功标志图标
      <block wx:for="{{iconSizeColor}}">
        <icon class="flex-row" type="safe_success" size="{{item.size}}" color=
" {{item.color}}" />
      </block>
    </view>

    <view>安全警告图标
      <block wx:for="{{iconSizeColor}}">
        <icon  class="flex-row"  type="safe_warn"  size="{{item.size}}"  color=
"{{item.color}}" />
      </block>
    </view>

    <view>带圆的成功图标
      <block wx:for="{{iconSizeColor}}">
        <icon   class="flex-row"   type="success_circle"   size="{{item.size}}"
color=" {{item.color}}" />
      </block>
    </view>

    <view>不带圆的成功图标
      <block wx:for="{{iconSizeColor}}">
        <icon  class="flex-row"  type="success_no_circle"  size="{{item.size}}"
color="{{item.color}}" />
      </block>
    </view>

    <view>带圆的等待图标
      <block wx:for="{{iconSizeColor}}">
        <icon class="flex-row" type="waiting_circle" size="{{item.size}}" color=
" {{item.color}}" />
      </block>
    </view>

    <view>圆圈图标
      <block wx:for="{{iconSizeColor}}">
        <icon   class="flex-row"   type="circle"   size="{{item.size}}"   color=
"{{item.color}}" />
      </block>
    </view>

    <view>下载图标
      <block wx:for="{{iconSizeColor}}">
        <icon    class="flex-row"    type="download"    size="{{item.size}}"
color="{{item.color}}" />
      </block>
    </view>
```

非卖品，仅供非商业用途或交流学习使用

```
    <view>带圆的提示信息图标
      <block wx:for="{{iconSizeColor}}">
        <icon class="flex-row" type="info_circle" size="{{item.size}}" color=
"{{item. color}}" />
      </block>
    </view>

    <view>取消图标
      <block wx:for="{{iconSizeColor}}">
        <icon  class="flex-row"  type="cancel"  size="{{item.size}}"  color=
"{{item. color}}" />
      </block>
    </view>

    <view>搜索图标
      <block wx:for="{{iconSizeColor}}">
        <icon  class="flex-row"  type="search"  size="{{item.size}}"  color=
"{{item.color}}" />
      </block>
    </view>

    <view>清除图标
      <block wx:for="{{iconSizeColor}}">
        <icon  class="flex-row"  type="clear"  size="{{item.size}}"  color=
"{{item.color}}" />
      </block>
    </view>

    // pages/iconPage/iconPage.js
    Page({
      data: {
        iconSizeColor: [{
          size: 20,
          color: 'red'
        }, {
          size: 30,
          color: 'orange'
        }, {
          size: 40,
          color: 'yellow'
        }, {
          size: 50,
          color: 'green'
        },
        {
          size: 60,
          color: 'rgb(0,255,255)'
        }]
      }
    })
```

3.3.2 text

text 文本组件，支持转义字符“\”。text 这个组件是唯一可以长按选中的文本。<text>

组件内只支持 <text> 嵌套。使用代码如下所示：

```
<text class="page_text">文本组件详细介绍</text>
```

文本的字号、颜色、字体等属性都是通过 CSS 样式来修改。

1）字体大小设置

给 text 组件设置属性 class="page_text"，找到页面对应的.wxss 文件配置，实现“page_text”样式，代码如下所示：

```
/* pages/textPage/textPage.wxss */
.page_text {
    font-size:30px;
}
```

font-size:30px; 就是设置字体大小的具体代码。

2）字体颜色设置

在.page_text 属性中，增加“color: #ff5400;”即可实现颜色修改，具体代码如下所示：

```
/* pages/textPage/textPage.wxss */
.page_text {
    font-size:30px;
    color: #ff5400;
}
```

3）字体样式设置

font-family 用来设置字体名称，如图 3-25 所示。font-style 用来设置文字样式，如图 3-26 所示。

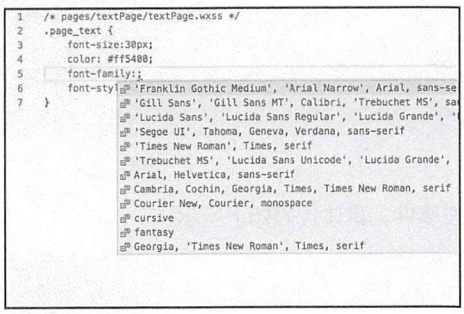


图 3-25 font-family 参数

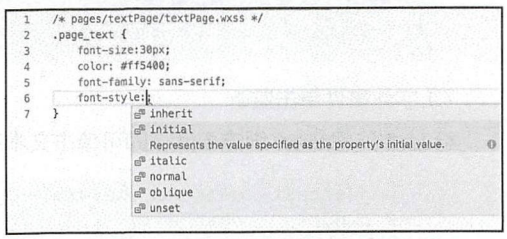


图 3-26 font-style 参数

4）控制字体换行显示

可以采用标签 <view></view> 将 text 组件包裹起来，作为父标签，这样 text 组件就可以在新的一行显示。或者采用设置 class 属性加上 display: block。代码如下所示：

```
<!--pages/textPage/textPage.wxml-->
<view>
 <text class="page_text">文本组件详细介绍</text>
</view>
 <view>
```

```
    <text class="page_text">第一章</text>
  </view>
  <view>
    <text class="page_text">第二章</text>
  </view>
```

5）控制字体距离周围边距

给 text 在 class 属性中根据需要采用 margin-top、margin-bottom、padding 等这类属性，具体代码如下所示：

```
/* pages/textPage/textPage.wxss */
.page_text {
  font-size: 10px;
  color: #ff5400;
  font-family: sans-serif;
  font-style: initial;
  margin-top: 1mm;
  margin-bottom: 1cm;
  padding: 1cm;
}
```

6）字体动态获取方式

使用“{{}}”形式，动态绑定数据，例如：<text>{{text}}</text>，这里将文本静态内容改成 {{text}}，然后在对应的 .js 文件 data 中赋值，具体代码如下所示：

```
<!--pages/textPage/textPage.wxml-->
<view>
  <text>{{text}}</text>
</view>

// pages/textPage/textPage.js
Page({
  data: {
    text: "这是动态绑定数据"
  }
})
```

7）字体事件单击监听

给 text 外层 view 绑定事件，即可单击文本触发事件，具体代码如下所示：

```
<!--pages/textPage/textPage.wxml-->
<view bindtap="toast">
  <text>{{text}}</text>
</view>

// pages/textPage/textPage.js
Page({
  data: {
    text: "这是动态绑定数据"
  },
  toast:function()
  {
console.log('text 单击事件');
  }
})
```

对 view 添加 bindtap 属性，这里 toast 可以自定义名称，然后在对应的 .js 文件中实现 toast:function()方法。

8）字体文本内容更改或显示

采用 this.setData() 即可更改或显示字体文本内容，this 代表当前组件对象，text 是文本名称，具体代码如下所示：

```
// pages/textPage/textPage.js
Page({
  data: {
    text: "这是动态绑定数据"
  },
  toast: function () {
    console.log('text 单击事件');
    this.setData
      (
      {
        text: "远程返回结果: "
      }
      )
  }
})
```

text 文本组件的属性，如表 3-10 所示。

表 3-10　　text 属性

属性名	类型	默认值	说明	最低版本
selectable	Boolean	false	文本是否可选	1.1.0
space	String	false	显示连续空格	1.4.0
decode	Boolean	false	是否解码	1.4.0

space 属性的有效值，如表 3-11 所示。

表 3-11　　space 有效值

值	说明
ensp	中文字符空格一半大小
emsp	中文字符空格大小
nbsp	根据字体设置的空格大小

decode 可以解析的有   < > & '   。各个操作系统的空格标准并不一致。<text/> 组件内只支持 <text/> 嵌套。除了文本节点以外的其他节点都无法被长按选中。

3.3.3 rich-text

从基础库 1.4.0 开始支持 rich-text 富文本组件。rich-text 支持默认事件，包括：tap、

touchstart、touchmove、touchcancel、touchend 和 longtap。其属性如表 3-12 所示。

表 3-12 rich-text 属性

属性	类型	默认值	说明	最低版本
nodes	Array / String	[]	节点列表 / HTML String	1.4.0

nodes 属性推荐使用 Array 类型，由于组件会将 String 类型转换为 Array 类型，所以 nodes 不推荐使用 String 类型，性能会有所下降。

nodes 现支持两种节点，通过 type 来区分，分别是元素节点和文本节点，默认是元素节点，在富文本区域里显示的是 HTML 节点。

元素节点：type = node，属性如表 3-13 所示。

表 3-13 type = node 属性

属性	说明	类型	必填	备注
name	标签名	String	是	支持部分受信任的 HTML 节点
attrs	属性	Object	否	支持部分受信任的属性，遵循 Pascal 命名法
children	子节点列表	Array	否	结构和 nodes 一致

文本节点：type = text，属性如表 3-14 所示。

表 3-14 type = text 属性

属性	说明	类型	必填	备注
text	文本	String	是	支持 entities

几点注意事项如下：

1. 受信任的 HTML 节点及属性，全局支持 class 和 style 属性，但不支持 id 属性。
2. rich-text 组件内屏蔽所有节点的事件。
3. name 属性对大小写不敏感。
4. 如果使用了不受信任的 HTML 节点，该节点及其所有子节点将会被移除。
5. img 标签仅支持网络图片。

关于富文本使用代码如下所示：

```
// pages/richTextPage/richTextPage.js
Page({

  data: {
    nodes: [{
      name: 'div',
      attrs: {
        class: 'div_class',
        style: 'line-height: 60px; color: red;'
      },
      children: [{
```

```
        type: 'text',
        text: 'Hello World!'
      }]
    }]
  },

  tap() {
    console.log('tap')
  }
})

<!--pages/richTextPage/richTextPage.wxml-->
<rich-text nodes="{{nodes}}" bindtap="tap"></rich-text>
```

3.3.4 progress

进度条描述的是一种加载的状态，例如图片下载的进度、软件升级下载的进度，等等。

进度条组件 progress 有 5 个属性，如表 3-15 所示。

表 3-15 progress 属性

属性名	类型	默认值
percent	Float	无
show-info	Boolean	false
stroke-width	Number	6
color	Color	#09BB07
active	Boolean	false

- percent：百分比，指示完成度。
- show-info：是否显示右侧的百分数字，有无值不重要，show-info="true"是同等效果。
- stroke-width：代表线条的宽度。
- color：颜色。
- active：表示有出场动画，需设定一个百分比值，动画显示的时候也是从 0 开始加载到设定的百分比。

具体使用代码如下所示：

```
<!--pages/progressPage/progressPage.wxml-->
<View >
  <!--百分比是 20,并在进度条右侧显示百分比-->
  <Text class="text-style">百分比是 30,并在进度条右侧显示百分比</Text>
  <progress percent="20" show-info />
  <!--百分比是 40,进度条线的宽度 12px-->
  <Text class="text-style">百分比是 40,进度条线的宽度 12px</Text>
  <progress percent="40" stroke-width="12" />
  <!--百分比是 60,进度条颜色:pink-->
  <Text class="text-style">百分比是 60,进度条颜色:pink</Text>
```

```
  <progress percent="60" color="pink" />
  <!--百分比是 80,进度条从左往右的动画-->
  <Text class="text-style">百分比是 80,进度条从左往右的动画</Text>
  <progress percent="80" show-info active />
</View>
```

利用数据绑定，可以动态调整进度条百分比，具体代码如下：

```
<!--pages/progressPage/progressPage.wxml-->
<View >
  <progress percent="{{percent}}" show-info/>
<button bindtap="tap"> 进度条+20 </button>
</View>

// pages/progressPage/progressPage.js
Page({
  data:{
    percent:1
  },
  tap: function() {
    this.data.percent =  this.data.percent + 5
    this.setData({
      percent: this.data.percent
    })

  },
})
```

3.4 表单组件

表单组件主要用于构建与用户交互的表单，主要包括：button、checkbox、form、input、label、picker、picker-view、radio、slider、switch、textarea。本节将对各个组件一一介绍。

3.4.1 button

按钮组件在项目中经常用到，其属性如表 3-16 所示。

表 3-16　　button 属性

属性名	类型	默认值	说明
size	String	default	有效值 default、mini
type	String	default	按钮的样式类型，有效值 primary、default、warn
plain	Boolean	false	按钮是否镂空，背景色透明
disabled	Boolean	false	是否禁用
loading	Boolean	false	名称前是否带 loading 图标
form-type	String	无	有效值：submit、reset，用于 <form/> 组件，单击分别会触发 submit/reset 事件

续表

属性名	类型	默认值	说明
open-type	String		微信开放能力（1.1.0）
hover-class	String	button-hover	指定按钮按下去的样式类。当 hover-class="none" 时，没有单击态效果
hover-start-time	Number	50	按住后多久出现单击态，单位为 ms
hover-stay-time	Number	400	手指松开后单击态保留时间，单位为 ms
session-from	String		open-type="contact"时有效，用户从该按钮进入会话时，开发者将收到带上本参数的事件推送。本参数可用于区分用户进入客服会话的来源（1.4.0）
bindgetuserinfo	Handler		open-type="getUserInfo"时有效，用户单击该按钮时，会返回获取到的用户信息，从返回参数的 detail 中获取到的值同 wx.getUserInfo（1.3.0）

1）size 和 type 属性

size 属性的默认值为 default，type 属性的默认值也为 default，如果值为 default 时，该属性可以省略。两者使用代码如下所示，显示效果图如图 3-27 所示。

```
<!--pages/viewPage1/viewPage1.wxml-->
<!--  size 和 type 属性-->
<view>size 属性为 default、 type 属性为 default</view>
<button type="default" size="default"> 下一步 </button>

<view class="content">size 属性为 mini、 type 属性为 default</view>
<button type="default" size="mini"> 下一步</button>


<view class="content">size 属性为 default、 type 属性为 primary</view>
<button type="primary" size="default"> 发  布</button>

<view class="content">size 属性为 mini、 type 属性为 primary</view>
<button type="primary" size="mini"> 发  布 </button>

<view class="content">size 属性为 default、 type 属性为 warn，用于需要慎重单击的
按钮</view>
<button type="warn" size="default"> 确定删除 </button>

<view class="content">size 属性为 mini、type 属性为 warn，用于需要慎重单击的按钮</view>
<button type="warn" size="mini"> 确定删除</button>
```

2）plain 属性

plain 属性设置按钮是否镂空。背景色透明，就是按钮背景透明，字体的颜色是原来按钮背景的颜色。使用代码如下所示，显示效果如图 3-28 所示。

```
<!--pages/viewPage1/viewPage1.wxml-->
<!--  plain 属性-->
```

```
    <view class="content">plain 属性为 true、size 属性为 default、 type 属性为
default</view>
    <button type="default" size="default" plain="true">下一步</button>

    <view class="content">plain 属性为 true、size 属性为 default、 type 属性为
primary</view>
    <button type="primary" size="default" plain="true">发 布</button>

    <view class="content">plain 属性为 true、size 属性为 default、 type 属性为
warn</view>
    <button type="warn" size="default" plain="true">确定删除</button>
```

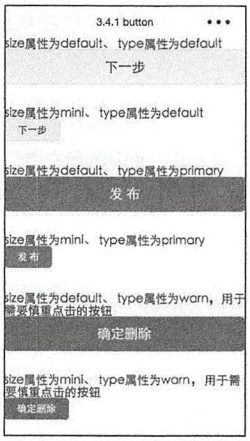


图 3-27　size 和 type 属性

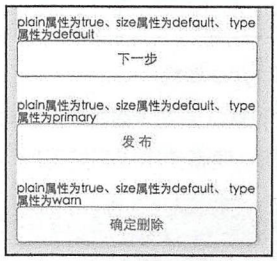


图 3-28　plain 属性

3）disable 属性

禁用按钮，默认等于 false，可以省略。如果设为 true，按钮禁用，不能单击。代码如下所示，显示效果如图 3-29 所示。

```
    <!-- disable 属性-->
    <view class="content">默认状态: "disabled = false"可以省略</view>
    <button type="primary" size="default" > size 属性为 default </button>

    <view class="content">disabled 属性为 true</view>
    <button type="primary" size="default" disabled="true"> size 属性为 mini </button>
```

4）loading 属性

loading 属性默认为 false，可以省略。如果设为 true，按钮的名称前带 loading 图标。使用代码如下所示，显示效果如图 3-30 所示。

```
    <!-- loading 属性-->
    <view class="content">loading 属性为 false 的时候可以省略</view>
    <button type="primary" size="default" > 没有 loading </button>

    <view class="content">loading 属性为 true</view>
    <button type="primary" size="default" loading="true" > 有 loading</button>
```

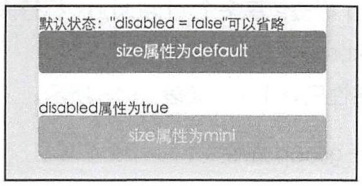


图 3-29 disable 属性

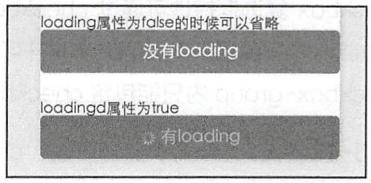


图 3-30 loading 属性

5）form-type 属性

form-type 属性设置按钮在表单中的类型，设置 submit、reset 值，在单击的时候会触发 submit/reset 事件，在 form 知识点中会讲解。

6）hover-class、hover-start-time、hover-stay-time 属性

hover-class 设置按钮按下去的样式，默认的样式是“{background-color:rgba(0,0,0,0.1); opacity:0.7;}”。hover-start-time 设置按住多久会触发单击态，单位为 ms。hover-stay-time 设置松手之后保留单击态的时间，单位为 ms。具体使用代码如下：

```
<!--pages/viewPage1/viewPage1.wxml-->
<!--  hover-class、hover-start-time、hover-stay-time 属性-->
<view class="content"></view>
//单击样式：other-button-hover，500ms 之后触发单击事件，松手之后单击效果保持 3000ms
<button type="default" size="default" bindtap="setDefault"
       hover-class="other-button-hover"  hover-start-time="500"  hover-stay-
time = "3000"> 修改单击样式 </button>
     /* pages/viewPage1/viewPage1.wxss */
     /*  hover-class、hover-start-time、hover-stay-time 属性*/
     /** 自定义 button 单击样式**/
     .other-button-hover {
       background-color: #ff5400;
       opacity: 0.7;
}
```

7）open-type 有效值，如表 3-17 所示。

表 3-17 open-type 有效值

值	说明	最低版本
contact	打开客服会话	1.1.0
share	触发用户转发，使用前建议先阅读使用指引	1.2.0
getUserInfo	获取用户信息，可以从 bindgetuserinfo 回调中获取到用户信息	1.3.0

使用代码如下所示：

```
<!--pages/viewPage1/viewPage1.wxml-->
<button open-type="contact">进入客服会话</button>
```

3.4.2 checkbox 与 checkbox-group

checkbox-group 是多项选择器，内部由多个 checkbox 组成。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

checkbox 复选框控件可通过 checkbox-group 标签包裹多个 checkbox 标签来实现多重选择功能，中间可以嵌入其他控件，并可在 checkbox-group 中设置监听事件。

checkbox-group 内只能包含 checkbox，其属性只有 1 个，如表 3-18 所示。

表 3-18　　checkbox-group 属性

属性名	类型	默认值	说明
bindchange	EventHandle		<checkbox-group/> 中选中项发生改变会触发 change 事件，detail = {value:[选中的 checkbox 的 value 数组]}

checkbox 组件为一个多选框被放到 checkbox-group 组中，其属性如表 3-19 所示。

表 3-19　　checkbox 属性

属性名	类型	默认值	说明
value	String		<checkbox/>标识，选中时触发<checkbox-group/> 的 change 事件，并携带 <checkbox/> 的 value
disabled	Boolean	false	是否禁用
checked	Boolean	false	当前是否选中，可用来设置默认选中
color	Color		checkbox 的颜色，同 CSS 的 color

disabled 禁止用户单击，checked 设置为 true 或 false，组件都会正常显示，只是不允许用户单击来改变 checked 属性。checked 使用代码如下所示，显示效果如图 3-31 所示。

```
    <!--pages/viewPage2/viewPage2.wxml-->
    <!--checkbox-group 有个监听事件 bindchange，监听数据选中和取消-->
    <checkbox-group bindchange="listenCheckboxChange">
    <!--这里用 view 显示内容，for 循环写法 wx:for-items 默认 item 为每一项-->
       <view style="display: flex;"  wx:for-items="{{items}}">
       <!--value 值和默认选中状态都是通过数据绑定在 js 中的-->
          <checkbox  color="#ff5400"  value="{{item.name}}"  checked="{{item.
checked}}"/ >{{item.value}}
       </view>
    </checkbox-group>

    // pages/viewPage2/viewPage2.js
    Page({
      data: {
        items: [
          {name: 'CHN', value: '中国', checked: 'true'},
          {name: 'USA', value: '美国'},
          {name: 'GBR', value: '英国'},
          {name: 'FRA', value: '法国'},
          {name: 'BRA', value: '巴西'},
          {name: 'JPN', value: '日本'},
        ]
      },
      checkboxChange: function(e) {
        console.log('checkbox 发生 change 事件，携带 value 值为: ', e.detail.value)
      },
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
  /**
   * 监听 checkbox 事件
   */
  listenCheckboxChange:function(e) {
      console.log('当 checkbox-group 中的 checkbox 选中或者取消时被调用');
      //打印对象包含的详细信息
      console.log(e);

  },
})
```

上例中有 6 个 checkbox 设置了 color 属性，勾选选项，checkbox 在单击的时候会自动触发“checkbox-Change”事件，.js 文字中“checkboxChange”事件，用来改变 checkbox 组件的 checked 属性值。checkbox-group 配置 bindchange="listenCheckboxChange"事件，当 checkbox-group 中的 checkbox 选中或者取消时被调用，每次选择或者取消选择 checkbox，都会触发 listenCheckboxChange 方法，然后打印出来所有选择的数据（即 e.detail.value）到控制台。

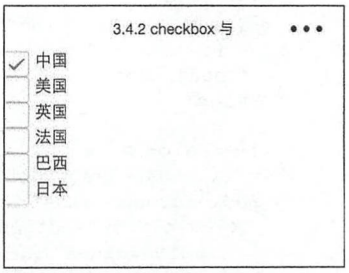


图 3-31 checked 效果

3.4.3 form

表单，将组件内用户输入的 switch、input、checkbox、slider、radio、picker 的 value 值进行提交。当单击 <form/> 表单中 formType 为 submit 的 <button/> 组件时，会将表单组件中的 value 值进行提交。每个 form 表单内的组件都必须有 name 属性来作为 key，否则提交不上去。button 中的 type 有两个属性 submit 和 reset，分别对应 form 的两个事件。form 组件的属性如表 3-20 所示。

表 3-20 form 属性

属性	类型	说明
report-submit	Boolean	是否返回 formId 用于发送模板消息
bindsubmit	EventHandle	携带 form 中的数据触发 submit 事件，event.detail = {value : {'name': 'value'} , formId: ''}
bindreset	EventHandle	表单重置时会触发 reset 事件

使用代码如下所示，显示效果如图 3-32 所示。

```
<!--pages/viewPage3/viewPage3.wxml-->
<form bindsubmit="formSubmit" bindreset="formReset">

  <!--switch-->
  <view class="section section_gap">
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    <view class="section__title">switch 开关</view>
    <switch name="switch" />
  </view>

  <!--slider-->
  <view class="section section_gap">
    <view class="section__title">slider 滑块</view>
    <slider name="slider" show-value></slider>
  </view>

  <!--input-->
  <view class="section">
    <view class="section__title">input 输入框</view>
    <input name="input" placeholder="please input here" />
  </view>

  <!--radio-->
  <view class="section">radio 单选</view>
  <radio-group name="radio-group">
    <view class="sectionMiNi" style="display: flex;">
      <radio value="radio1" />radio1</view>
    <view class="sectionMiNi" style="display: flex;">
      <radio value="radio2" />radio2</view>
  </radio-group>

  <!--checkbox-->
  <view class="section">checkbox 多选</view>
  <checkbox-group name="checkbox">
    <view class="sectionMiNi" style="display: flex;">
      <checkbox value="checkbox1" />checkbox1</view>
    <view class="sectionMiNi" style="display: flex;">
      <checkbox value="checkbox2" />checkbox2</view>
  </checkbox-group>

  <!--按钮-->
  <view class="btn-area">
    <button formType="submit">Submit</button>
    <button formType="reset">Reset</button>
  </view>
</form>

/* pages/viewPage3/viewPage3.wxss */
.section {
  margin-top: 20px;
}

.sectionMiNi {
  margin-top: 6px;
}

// pages/viewPage3/viewPage3.js
Page({
  formSubmit: function(e) {
    console.log('form 发生了 submit 事件，携带数据为: ', e.detail.value)
  },
  formReset: function() {
    console.log('form 发生了 reset 事件')
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    }
})
```

单击 Submit 按钮，会触发.js 文件中的“formSubmit”事件。formSubmit 事件写了 log 输出，控制台会输出如图 3-33 所示的内容。

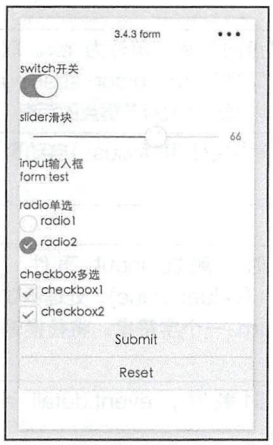


图 3-32 form 效果

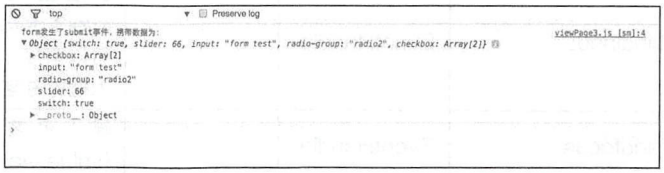


图 3-33 Submit 事件输出 form 所有控件 value

页面的 <form/> 组件，如果属性 report-submit 为 true 时，可以声明为需发模板消息，此时单击按钮提交表单可以获取 formId，用于发送模板消息。或者当用户完成支付行为，可以获取 prepay_id 用于发送模板消息。调用接口下发模板消息，具体可以参考第五章 5.6 模板消息。

3.4.4 input

input 输入框使用的频率比较高，例如：登录、注册、获取搜索框中的内容等都需要用到，设置 input 组件样式可以在 input 外面包裹 view 组件，实现自定义样式和布局。input 属性如表 3-21 所示。

表 3-21 input 属性

属性	类型	默认值	说明
value	String		输入框的初始内容
type	String	text	input 的类型，有效值：text、number、idcard、digit
password	Boolean	false	是否是密码类型
placeholder	String		输入框为空时占位符
placeholder-style	String		指定 placeholder 的样式
placeholder-class	String	input-placeholder	指定 placeholder 的样式类

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

续表

属性	类型	默认值	说明
disabled	Boolean	false	是否禁用
maxlength	Number	140	最大输入长度，设置为-1 的时候不限制最大长度
cursor-spacing	Number	0	指定光标与键盘的距离，单位为 px。取 input 距离底部的距离和 cursor-spacing 指定的距离的最小值作为光标与键盘的距离
auto-focus	Boolean	false	(即将废弃，请直接使用 focus)自动聚焦，拉起键盘
focus	Boolean	false	获取焦点
bindinput	EventHandle		当键盘输入时，触发 input 事件，event.detail = {value: value}，处理函数可以直接 return 一个字符串，将替换输入框的内容
bindfocus	EventHandle		输入框聚焦时触发，event.detail = {value: value}
bindblur	EventHandle		输入框失去焦点时触发，event.detail = {value: value}
bindconfirm	EventHandle		单击完成按钮时触发，event.detail = {value: value}

value：输入框默认显示的内容，代码示例如下：

```
<input class="section10" placeholder="请输入账号" value="185****8888"/>
```

type：设置键盘的样式，有 4 种类型：text（字母键盘）、number（数字键盘）、idcard（身份证输入键盘）、digit（带小数点的数字键盘）。

password：设置文本框显示密码样式，代码如下所示：

```
<input class="section10" placeholder="请输入密码" password focus="{{focus}}" />
```

placeholder：占位字符，文本框没有内容时显示的内容。

placeholder-style：设置占位符的样式，代码如下所示：

```
<input placeholder-style="color:red" placeholder="占位符字体是红色的" />
```

placeholder-class：设置占位符 CSS 样式类名，代码如下所示：

```
<input type="idcard" placeholder-class="placeholderClass" placeholder="身份证输入键盘" />
```

disabled：禁用属性，设置后，不能单击文本框，代码如下所示：

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    <input type="idcard"  disabled placeholder="文本框禁用" />
```

maxlength：设置最大输入长度，设置为 -1（默认值）时不限制最大长度，代码如下所示：

```
    <input  maxlength="10" placeholder="最多只允许输入 10 个字符" />
```

cursor-spacing：指定光标与键盘的距离，单位为 px。取 input 距离底部的距离和 cursor-spacing 指定的距离的最小值作为光标与键盘的距离，代码如下所示：

```
    <input class="section10"placeholder="请输入账号"value="185****8888"
cursor -spacing = "20px"/>
```

auto-focus：设置该属性，默认光标定位到该文本框，代码如下所示：

```
    <input class="section10" placeholder="请输入账号" auto-focus/>
```

focus：设置焦点，光标移到该文本框，代码如下所示：

```
    <input class="section10" placeholder="请输入密码" password focus="{{focus}}" />
    <button bindtap="bindButtonTap">下一步</button>

    // pages/viewPage4/viewPage4.js
    Page({
     data: {
       focus: false,
      },
      bindButtonTap: function () {
        this.setData({
          focus: true
        })
      }
    })
```

4 个绑定事件的属性，分别是：

bindinput：绑定事件，键盘输入文本时会触发该事件。

bindfocus：输入框聚焦时触发。

bindblur：输入框失去焦点时触发。

bindconfirm：单击完成按钮时触发。

使用代码如下所示：

```
    <!--pages/viewPage4/viewPage4.wxml-->
    <input class="section10"placeholder="请输入账号"value="185****8888"
cursor-spacing="20px"bindinput="bindinputT"bindfocus="bindfocusT"bindblur="bindblurT"
bindconfirm="bindconfirmT"/>

    // pages/viewPage4/viewPage4.js
    Page({
      data: {
        focus: false,
        inputValue: '',
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    phone: '',
    password: '',
  },

  // 绑定事件，键盘输入文本时会触发该事件
  bindinputT: function (e) {
    console.log(e.detail.value);
  },
  // 输入框聚焦时触发
  bindfocusT: function (e) {
    console.log(e.detail.value);
  },

  // 输入框失去焦点时触发
  bindblurT: function (e) {
    console.log(e.detail.value);
  },

  // 单击完成按钮时触发
  bindconfirmT: function (e) {
    console.log(e.detail.value);
  }
})
```

更多示例代码，如下所示：

```
<!--pages/viewPage4/viewPage4.wxml-->

<view class="section20">默认显示内容
<input class="section10"placeholder="请输入账号"alue="185****8888"
cursor-spacing="20px"bindinput="bindinputT"bindfocus="bindfocusT"
bindblur="bindblurT" bindconfirm="bindconfirmT"/>
</view>


<view class="section20">设置 auto-focus 属性，打开界面光标自动定位到此文本框
  <input class="section10" placeholder="请输入账号" auto-focus/>
</view>

<view class="section20">focus 值动态绑定，单击按钮修改 “focus=true”，光标定位到
该文本框
  <input class="section10" placeholder="请输入密码" password focus="{{focus}}" />
    <button bindtap="bindButtonTap">下一步</button>
</view>

<view class="section20">
  <input  maxlength="10" placeholder="最多只允许输入 10 个字符" />
</view>

<view class="section20">
  <input  bindinput="bindKeyInput" placeholder="输入的内容在下面 view 显示"/>
  <view class="section__title">你输入的是：{{inputValue}}</view>
</view>

<view class="section20">
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    <input bindinput="bindReplaceInput" placeholder="连续的三个 1 会变成 3" />
  </view>

  <view class="section20">
    <input bindinput="bindHideKeyboard" placeholder="输入 123 自动收起键盘" />
  </view>

  <view class="section20">
    <input password placeholder="数字键盘" type="number" />
  </view>

  <view class="section20">
    <input password placeholder="字母键盘" type="text" />
  </view>

  <view class="section20">
    <input type="digit" placeholder="带小数点的数字键盘"/>
  </view>

  <view class="section20">
    <input type="idcard" placeholder-class="placeholderClass" placeholder="身
份证输入键盘" />
  </view>

  <view class="section20">
    <input type="idcard" disabled placeholder="文本框禁用" />
  </view>

  <view class="section20">
    <input placeholder-style="color:red" placeholder="占位符字体是红色的" />
  </view>

  <!--例子，使用 input 实现密码框-->
  <!--style 的优先级比 class 高会覆盖和 class 相同的属性-->
  <view class="inputView" style="margin-top: 40% ">
  <input class="input"type="number"placeholder="请输入账号"placeholder-style
="color: red" bindinput="listenerPhoneInput" />
  </view>

  <view class="inputView">
  <input class="input" password="true" placeholder="请输入密码" placeholder-
style="color: red" bindinput="listenerPasswordInput"/>
  </view>
  <button style="margin-left: 16px; margin-right: 16px; margin-top: 50rpx;
margin-bottom:40px;border-radius: 40rpx" type="primary" bindtap="listenerLogin">登
录</button>


  // pages/viewPage4/viewPage4.js
  Page({
    data: {
      focus: false,
      inputValue: '',
      phone: '',
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    password: '',
  },

  // 绑定事件，键盘输入文本时会触发该事件
  bindinputT: function (e) {
    console.log(e.detail.value);
  },
  // 输入框聚焦时触发
  bindfocusT: function (e) {
    console.log(e.detail.value);
  },

  // 输入框失去焦点时触发
  bindblurT: function (e) {
    console.log(e.detail.value);
  },

  // 单击完成按钮时触发
  bindconfirmT: function (e) {
    console.log(e.detail.value);
  },


  bindButtonTap: function () {
    this.setData({
      focus: true
    })
  },
  bindKeyInput: function (e) {
    this.setData({
      inputValue: e.detail.value
    })
  },
  bindReplaceInput: function (e) {
    var value = e.detail.value
    var pos = e.detail.cursor
    if (pos != -1) {
      //光标在中间
      var left = e.detail.value.slice(0, pos)
      //计算光标的位置
      pos = left.replace(/111/g, '3').length
    }

    //直接返回对象，可以对输入进行过滤处理，同时可以控制光标的位置
    return {
      value: value.replace(/111/g, '3'),
      cursor: pos
    }
  },
  bindHideKeyboard: function (e) {
    if (e.detail.value === '123') {
      //收起键盘
      wx.hideKeyboard()
    }
```

非卖品，仅供非商业用途或交流学习使用

```
  },
  /**
  * 监听手机号输入
  */
  listenerPhoneInput: function (e) {
    this.data.phone = e.detail.value;

  },

  /**
   * 监听密码输入
   */
  listenerPasswordInput: function (e) {
    this.data.password = e.detail.value;
  },

  /**
   * 监听登录按钮
   */
  listenerLogin: function () {
    //打印输入账号和密码
    console.log('手机号为: ', this.data.phone);
    console.log('密码为: ', this.data.password);
  },

})

/* pages/viewPage4/viewPage4.wxss */
.section20 {
  margin-top: 20px;
}
.section10 {
  margin-top: 10px;
}

.input{
  padding-left: 16px;
  height: 45px;
}

.inputView{
  border: 2px solid red;
  border-radius: 40px;
  margin-left: 16px;
  margin-right: 16px;
  margin-top: 16px;
}

.placeholderClass{
  size: 18px;
  color:red;
}
```

以上代码，显示效果如图 3-34 所示。

非卖品，仅供非商业用途或交流学习使用

图 3-34　input 控件使用

3.4.5　label

标签组件在小程序中被用在了很多地方，主要用于为另一个组件提供说明性的文本。在小程序中，只有一个 for 属性指示另一个组件的 id，可用来改进表单组件的可用性，使用 for 属性找到对应的 id，或者将控件放在该标签下，当单击时，就会触发对应的控件。for 优先级高于内部控件，内部有多个控件的时候默认触发第一个控件。

目前可以绑定的控件有：button、checkbox、radio、switch。

label 标签属性，如表 3-22 所示。

表 3-22　　label 属性

属性	类型
for	String

label 组件有两种使用方式，一种是表单组件在 label 内，另一种是 label 用 for 标识表单

组件。具体代码如下所示：

```
    <!--pages/viewPage5/viewPage5.wxml-->
    <view >表单组件在 label 内</view>
    <checkbox-group bindchange="listenCheckboxChange">
    <!--这里用 view 显示内容，for 循环写法 wx:for-items 默认 item 为每一项-->
       <label style="display: flex;"  wx:for-items="{{items}}">
       <!--value 值和默认选中状态都是通过数据绑定在 js 中的-->
          <checkbox  color="#ff5400"  value="{{item.name}}"  checked="{{item.
checked}}"/> {{item.value}}
       </label>
    </checkbox-group>



    <view class="section20">label 用 for 标识表单组件</view>
    <radio-group bindchange="radioChange">
      <view class="label" wx:for="{{radioItems}}">
       <radio id="{{item.name}}" hidden value="{{item.name}}" checked="{{item.
checked}}"></radio>
       <view class="label__icon">
         <view class="label__icon-checked"style="opacity:{{item.checked ? 1:
0}}"></view>
       </view>
       <label   class="label__text"   for="{{item.name}}"><text>{{item.name}}
</text></label>
      </view>
    </radio-group>


    // pages/viewPage5/viewPage5.js
    Page({
      data: {
        items: [
          {name: 'USA', value: '美国'},
          {name: 'CHN', value: '中国', checked: 'true'},
          {name: 'BRA', value: '巴西'},
          {name: 'JPN', value: '日本', checked: 'true'},
          {name: 'GBR', value: '英国'},
          {name: 'FRA', value: '法国'},
        ],
        radioItems: [
          {name: 'USA', value: '美国'},
          {name: 'CHN', value: '中国', checked: 'true'},
          {name: 'BRA', value: '巴西'},
          {name: 'JPN', value: '日本'},
          {name: 'GBR', value: '英国'},
          {name: 'FRA', value: '法国'},
        ],
        hidden: false
      },
      checkboxChange: function(e) {
        var checked = e.detail.value
        var changed = {}
        for (var i = 0; i < this.data.checkboxItems.length; i ++) {
          if (checked.indexOf(this.data.checkboxItems[i].name) !== -1) {
```

```
        changed['checkboxItems['+i+'].checked'] = true
      } else {
        changed['checkboxItems['+i+'].checked'] = false
      }
    }
    this.setData(changed)
  },
  radioChange: function(e) {
    var checked = e.detail.value
    var changed = {}
    for (var i = 0; i < this.data.radioItems.length; i ++) {
      if (checked.indexOf(this.data.radioItems[i].name) !== -1) {
        changed['radioItems['+i+'].checked'] = true
      } else {
        changed['radioItems['+i+'].checked'] = false
      }
    }
    this.setData(changed)
  }
})

/* pages/viewPage5/viewPage5.wxss */
.section20 {
  margin-top: 20px;
}

/*label 样式*/
.label{
   margin-bottom: 16px;
}
/*label 文本的样式*/
.label__text {
   display: inline-block;
   vertical-align: middle;
}

/*默认 icon 样式*/
.label__icon {
   position: relative;
   display: inline-block;
   vertical-align: middle;
   margin-right: 16px;
   width: 16px;
   height: 16px;
   background: #ffffff;
   border-radius: 50px;
}

/*选中的 icon 样式*/
.label__icon-checked {
   position: absolute;
   vertical-align: middle;
   left: 3px;
   top: 3px;
   width: 16px;
   height: 16px;
   background: #1aad19;
```

```
        border-radius: 50%;
    }
```

如果选择性控件不放在 label 内部，则需要使用 for 属性，for 属性与 radio 的 id 属性是同一个值。如果 label 将多个组件圈住，label 本身相当于是一个空的 block，如果此时 label 标签内部的第 2 个选择组件 radio 被 label 选中，text 用于提供显示的文本，当单击文本时，radio 同步是否选中的状态。

第 2 组中，使用样式 label__icon 来设置背景，大小是 16px × 16px，label__icon-checked 是选中之后的圆点 UI，上面和左边各有 2px 的偏离，宽高是 14px，合起来正好是 16px。运行效果如图 3-35 所示。

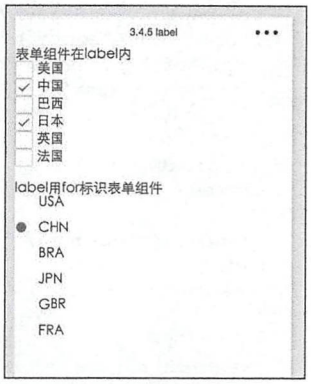


图 3-35 label 控件使用

3.4.6 picker

我们非常熟悉 select 下拉框，小程序对应的下拉框就是 picker。从底部弹出的滚动选择器，目前支持 3 种选择器，通过 mode 来区分，分别是普通选择器、时间选择器、日期选择器，默认是普通选择器。

- 普通选择器：mode=selector，属性如表 3-23 所示。

表 3-23 普通选择器属性

属性	类型	默认值	说明
range	Array / Object Array	[]（注：空数组）	mode 为 selector 时，range 有效
range-key	String		当 range 是一个 Object Array 时，通过 range-key 来指定 Object 中 key 的值作为选择器显示内容
value	Number	0	value 的值表示选择了 range 中的第几个（下标从 0 开始）
bindchange	EventHandle		value 改变时触发 change 事件，event.detail = {value: value}
disabled	Boolean	false	是否禁用

普通选择器自带 3 个必要属性：

1. range 主要指服务端返回的数据集（也可自定义数据），类型为 Array。

2. value 主要指 range 数组的下标，类型为 Number。

3. bindchange 主要指改变选项时触发的事件。其中 event.detail.value 可获取自身 value 的值。由于 range 属性值为整个下拉选项的数据集，value 为 range 的下标，所以要显示当前选项即：{{array[index]}}。

普通选择器，使用代码如下所示：

```
<!--pages/viewPage6/viewPage6.wxml-->
<view class="section">地区选择器
  <picker bindchange="bindPickerChange" value="{{index}}" range="{{array}}">
    <view class="picker">
      当前选择: {{array[index]}}
    </view>
  </picker>
</view>

/* pages/viewPage6/viewPage6.wxss */
page{
  background-color: #fbf9fe;
}

.section {
  margin-top: 20px;
}
.picker {
  background-color: #ffffff;
  height: 45px;
  display: flex;
  align-items: center;
}
.picker1 {
  background-color: #ffffff;
  width: 80px;
  height: 45px;
  display: flex;
  align-items: center;
}

// pages/viewPage6/viewPage6.js
Page({
    data: {
        array: ['中国', '美国', '巴西', '德国'],
        objectArray: [
            {
                id: 0,
                name: '中国'
            },
            {
                id: 1,
                name: '美国'
            },
            {
                id: 2,
                name: '巴西'
            },
            {
                id: 3,
                name: '日本'
            }
        ],
        index: 0,
```

```
        date: '2017-01-06',
        time: '12:00',
        province: '请选择',
        city: '请选择',
        county: '请选择',
    },
    bindPickerChange: function (e) {
        console.log('picker 发送选择改变，携带值为', e.detail.value)
        this.setData({
            index: e.detail.value
        })
    },
    bindDateChange: function (e) {
        this.setData({
            date: e.detail.value
        })
    },
    bindTimeChange: function (e) {
        this.setData({
            time: e.detail.value
        })
    },
    selectPCCchange: function () {

    }
})
```

● 时间选择器：mode=time（注：当 picker 默认值（value）的分钟大于 end 属性规定的时间时，出现可选择大于规定时间的 bug），属性如表 3-24 所示。

表 3-24　　时间选择器属性

属性	类型	默认值	说明
value	String		表示选中的时间，格式为"hh:mm"
start	String		表示有效时间范围的开始，字符串格式为"hh:mm"
end	String		表示有效时间范围的结束，字符串格式为"hh:mm"
bindchange	EventHandle		value 改变时触发 change 事件，event.detail = {value: value}
disabled	Boolean	false	是否禁用

时间选择器，使用代码如下所示：

```
<!--pages/viewPage6/viewPage6.wxml-->
<view class="section">时间选择器
  <picker mode="time" value="{{time}}" start="09:01" end="21:01" bindchange=
"bindTimeChange">
    <view class="picker">
      当前选择：{{time}}
    </view>
  </picker>
</view>
```

● 日期选择器：mode = date，属性如表 3-25 所示。

表 3-25　　日期选择器属性

属性	类型	默认值	说明
value	String	0	表示选中的日期，格式为"YYYY-MM-DD"
start	String		表示有效日期范围的开始，字符串格式为"YYYY-MM-DD"
end	String		表示有效日期范围的结束，字符串格式为"YYYY-MM-DD"
fields	String	day	有效值 year、month、day，表示选择器的粒度
bindchange	EventHandle		value 改变时触发 change 事件，event.detail = {value: value}
disabled	Boolean	false	是否禁用

日期选择器使用代码如下所示：

```
    <view class="section">日期选择器
      <picker mode="date" value="{{date}}" start="2015-09-01" end="2017-09-
01" bindchange="bindDateChange">
        <view class="picker">
          当前选择: {{date}}
        </view>
      </picker>
    </view>
```

下面自定义 3 级联动，来选择省市县，代码如下所示，显示效果如图 3-36 所示。

```
    <!--3 级联动  省市县-->
    <view class="section">选择省市县
      <picker mode="selector" bindchange="bindPickerChangeP" value="{{index}}"
range="{{pccArray}}" value="{{pccIndex}}">
        <view class="picker1 {{hidePPicker?'hide':''}}" >{{province}}</view>
      </picker>

      <picker mode="selector" bindchange="bindPickerChangeCity" value="{{index}}"
range="{{ccArray}}" value="{{ccIndex}}">
        <view class="picker1 {{hideCCPicker?'hide':''}}">{{city}}</view>
      </picker>

      <picker mode="selector" bindchange="bindPickerChangeCounty" value="{{index}}"
range="{{cArray}}" value="{{cIndex}}">
        <view class="picker1 {{hideCOPicker?'hide':''}}">{{county}}</view>
      </picker>
    </view>
```

图 3-36　选择省市县

实现思路：

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

1. 在页面加载后，解析获得省份数组 pccArray，代码如下所示：

```
onReady: function (options) {
        // 页面初始化，options 为页面跳转所带来的参数

        // 获取省份，取数组中对象的 name 值，就是省份名称
        var tempArray = []; //临时数组，暂存省份

        //遍历省份，获得省份名称
        for (var i in pcc) {
            tempArray.push(pcc[i].name)
            this.setData({
                pccArray: tempArray
            })
        }

        //隐藏城市和县城的选择
        this.setData({
            hideCCPicker: true,
            hideCOPicker: true,
            province: '请选择省',
            city: '请选择市',
            county: '请选择县',
        })
    },
```

2. 选择省份之后，触发“bindPickerChangeP”事件，在该事件下，解析选择的省份和该省份下所有的城市，具体代码如下所示：

```
// 选择省份
    bindPickerChangeP: function (e) {

        //选择省份的 index
        pccIndex = e.detail.value

        //获取选择省份的名称
        this.setData({
            province: pcc[pccIndex].name
        })

        //获取该省下面所有的城市
        var array = pcc[pccIndex].sub

        //判断如果城市数量=0，隐藏城市和县城的选择
        if (array.length > 0) {
            this.setData({
                hideCCPicker: false,
                city: '请选择市',
                county: '请选择县',
            })
        } else {
            this.setData({
                //隐藏城市选择、县城选择
                hideCCPicker: true,
                hideCOPicker: true
            })
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        }

        var tempArray = []; //临时数组，暂存城市

        //遍历城市，获得城市名称
        for (var i in array) {

            tempArray.push(array[i].name)
            this.setData({
                ccArray: tempArray
            })
        }
    },
```

3. 选择城市之后，触发“bindPickerChangeCity”，在该事件下，获得所选城市的名称和该城市下所有的县城，代码如下所示：

```
// 选择城市
    bindPickerChangeCity: function (e) {

        //选择城市的 index
        ccIndex = e.detail.value

        //根据省份的 index，获取所有城市
        var array1 = pcc[pccIndex].sub

        //获取选择城市的名称
        this.setData({
            city: array1[ccIndex].name
        })

        //根据城市，获得所有县城
        var array = array1[ccIndex].sub

        //判断是否存在县城
        if (array != undefined) {
            this.setData({
                hideCOPicker: false,
                county: '请选择县'
            })
        } else {
            this.setData({
                hideCOPicker: true
            })
        }

        console.log(array)

        var tempArray = []; //临时数组，暂存县城名称

        //遍历所有县城，获得县城名称
        for (var i in array) {


            tempArray.push(array[i].name)
            this.setData({
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
                cArray: tempArray
            })
        }
    },
```

4. 选择县城后，触发“bindPickerChangeCounty”事件，在该事件下获取所选的县城名称，代码如下所示：

```
// 选择县城
    bindPickerChangeCounty: function (e) {

        //选择县城的 index
        cIndex = e.detail.value

        //根据省份的 index，获取所有城市
        var array1 = pcc[pccIndex].sub

        //根据城市，获得所有县城
        var array = array1[ccIndex].sub

        //获取选择县城的名称
        this.setData({
            county: array[cIndex].name
        })

        console.log(e.detail.value)
    }
```

- 多列选择器：mode = multiSelector（最低版本：1.4.0）。

从版本 1.4.0 之后，picker 增加多列选择器功能，其属性如表 3-26 所示。

表 3-26　　picker 多列选择器属性

属性	类型	默认值	说明
range	二维 Array / 二维 Object Array	[]	mode 为 selector 或 multiSelector 时，range 有效。二维数组，长度表示多少列，数组的每项表示每列的数据，如[["a","b"], ["c","d"]]
range-key	String		当 range 是一个二维 Object Array 时，通过 range-key 来指定 Object 中 key 的值作为选择器显示内容
value	Array	[]	value 每一项的值表示选择了 range 对应项中的第几个（下标从 0 开始）
bindchange	EventHandle		value 改变时触发 change 事件，event.detail = {value: value}
bindcolumn-change	EventHandle		某一列的值改变时触发 columnchange 事件，event.detail = {column: column, value: value}，column 的值表示改变了第几列（下标从 0 开始），value 的值表示变更值的下标
disabled	Boolean	false	是否禁用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

使用代码如下所示，运行效果参考源代码 Ch3.4。

```
<!--pages/viewPage6/viewPage6.wxml-->
<!-- 多列选择器 -->
<view class="section">
  <view class="section__title">多列选择器</view>
  <picker mode="multiSelector" bindchange="bindMultiPickerChange" bindcolumnchange="bindMultiPickerColumnChange" value="{{multiIndex}}" range="{{multiArray}}">
    <view class="picker">
      当前选择: {{multiArray[0][multiIndex[0]]}}, {{multiArray[1][multiIndex[1]]}}, {{multiArray[2][multiIndex[2]]}}
    </view>
  </picker>
</view>

// pages/viewPage6/viewPage6.js
//在该文件 Page data:下面定义:
multiArray: [['无脊椎动物', '脊椎动物'], ['扁形动物', '线形动物', '环节动物', '软体动物', '节肢动物'], ['猪肉绦虫', '吸血虫']],
        objectMultiArray: [
          [
            {
              id: 0,
              name: '无脊椎动物'
            },
            {
              id: 1,
              name: '脊椎动物'
            }
          ], [
            {
              id: 0,
              name: '扁形动物'
            },
            {
              id: 1,
              name: '线形动物'
            },
            {
              id: 2,
              name: '环节动物'
            },
            {
              id: 3,
              name: '软体动物'
            },
            {
              id: 3,
              name: '节肢动物'
            }
          ], [
            {
              id: 0,
              name: '猪肉绦虫'
            },
            {
              id: 1,
              name: '吸血虫'
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
            }
          ]
        ],
        multiIndex: [0, 0, 0] //默认选择 0, 0, 0

  //实现事件方法
  // 多列选择器
    bindMultiPickerChange: function (e) {
      console.log('picker 发送选择改变，携带值为', e.detail.value)
      this.setData({
        multiIndex: e.detail.value
      })
    },
    bindMultiPickerColumnChange: function (e) {
      console.log('修改的列为', e.detail.column, '，值为', e.detail.value);
      var data = {
        multiArray: this.data.multiArray,
        multiIndex: this.data.multiIndex
      };
      data.multiIndex[e.detail.column] = e.detail.value;
      switch (e.detail.column) {
        case 0:
          switch (data.multiIndex[0]) {
            case 0:
              data.multiArray[1] = ['扁形动物', '线形动物', '环节动物', '软体动
物', '节肢动物'];
              data.multiArray[2] = ['猪肉绦虫', '吸血虫'];
              break;
            case 1:
              data.multiArray[1] = ['鱼', '两栖动物', '爬行动物'];
              data.multiArray[2] = ['鲫鱼', '带鱼'];
              break;
          }
          data.multiIndex[1] = 0;
          data.multiIndex[2] = 0;
          break;
        case 1:
          switch (data.multiIndex[0]) {
            case 0:
              switch (data.multiIndex[1]) {
                case 0:
                  data.multiArray[2] = ['猪肉绦虫', '吸血虫'];
                  break;
                case 1:
                  data.multiArray[2] = ['蛔虫'];
                  break;
                case 2:
                  data.multiArray[2] = ['蚂蚁', '蚂蟥'];
                  break;
                case 3:
                  data.multiArray[2] = ['河蚌', '蜗牛', '蛞蝓'];
                  break;
                case 4:
                  data.multiArray[2] = ['昆虫', '甲壳动物', '蛛形动物', '多足动物'];
                  break;
              }
              break;
            case 1:
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
            switch (data.multiIndex[1]) {
              case 0:
                data.multiArray[2] = ['鲫鱼', '带鱼'];
                break;
              case 1:
                data.multiArray[2] = ['青蛙', '娃娃鱼'];
                break;
              case 2:
                data.multiArray[2] = ['蜥蜴', '龟', '壁虎'];
                break;
            }
            break;
        }
        data.multiIndex[2] = 0;
        console.log(data.multiIndex);
        break;
    }
    this.setData(data);
  },
  bindDateChange: function (e) {
    console.log('picker 发送选择改变，携带值为', e.detail.value)
    this.setData({
      date: e.detail.value
    })
  },
  bindTimeChange: function (e) {
    console.log('picker 发送选择改变，携带值为', e.detail.value)
    this.setData({
      time: e.detail.value
    })
  },
  bindRegionChange: function (e) {
    console.log('picker 发送选择改变，携带值为', e.detail.value)
    this.setData({
      region: e.detail.value
    })
  }
```

3.4.7 picker-view

嵌入页面的滚动选择器，picker-view 中只能放置<picker-view-column>组件，其他节点不会显示。节点的高度会自动设置成与 picker-view 选中框的高度一致。

picker-view 属性如表 3-27 所示。

表 3-27　　picker-view 属性

属性	类型	默认值	说明
value	Number Array		数组中的数字表示 picker-view 内的 picker-view-column 选择的第几项（下标从 0 开始），数字大于 picker-view-column 可选项长度时，选择最后一项
indicator-style	String		设置选择器中间选中框的样式

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

续表

属性	类型	默认值	说明
indicator-class	String		设置选择器中间选中框的类名
bindchange	EventHandle		当滚动选择，value 改变时触发 change 事件，event.detail = {value: value}；value 为数组，表示 picker-view 内的 picker-view-column 当前选择的是第几项（下标从 0 开始）

value：表示 picker-view 内的 picker-view-colume 选择的第几项（下标从 0 开始），数字大于 picker-view-column 可选项长度时，选择最后一项。

indicator-style：设置选择器中间选中框的样式。

bindchange：绑定事件，当滚动选择，value 改变时触发该事件。

下面以时间选择器为例，介绍一下 picker-view 的使用，首先需要选自己生成数据（年、月、日）供 picker-view 显示使用，代码如下所示：

```
// pages/viewPage7/viewPage7.js
// 自己生成数据：1990~2017 年
const date = new Date()
const years = []
for (let i = 1990; i <= date.getFullYear(); i++) {
  years.push(i)
}
// 自己生成数据：1~12 月
const months = []
for (let i = 1; i <= 12; i++) {
  months.push(i)
}

// 自己生成数据：1~31 天
const days = []
for (let i = 1; i <= 31; i++) {
  days.push(i)
}
```

这里初始化天数是 31 天，因为默认[years.length - 1, 1, 1]，代表 1 月有 31 天，选择器值改变的时候触发 bindChange 事件，此时会根据选中的月份重新计算该月的天数，具体代码如下所示，显示效果如图 3-37 所示。

图 3-37 picker-view 选择器

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    <!--pages/viewPage7/viewPage7.wxml-->
    <view>
      <view>{{year}}年{{month}}月{{day}}日</view>
      <picker-view  indicator-style="height:  50px;"  style="width:  100%;
height: 300px;" value="{{value}}" bindchange="bindChange">
        <picker-view-column>
          <view wx:for="{{years}}" style="line-height: 50px">{{item}}年</view>
        </picker-view-column>
        <picker-view-column>
          <view wx:for="{{months}}" style="line-height: 50px">{{item}}月</view>
        </picker-view-column>
        <picker-view-column>
          <view wx:for="{{days}}" style="line-height: 50px">{{item}}日</view>
        </picker-view-column>
      </picker-view>
    </view>

    // pages/viewPage7/viewPage7.js


    // 自己生成数据：1990~2017年
    const date = new Date()
    const years = []
    for (let i = 1990; i <= date.getFullYear(); i++) {
      years.push(i)
    }
    // 自己生成数据：1~12月
    const months = []
    for (let i = 1; i <= 12; i++) {
      months.push(i)
    }

    // 自己生成数据：1~31天
    const days = []
    for (let i = 1; i <= 31; i++) {
      days.push(i)
    }


    Page({
      data: {
        years: years, //picker-view第1列使用的数组
        months: months,//picker-view第2列使用的数组
        days: days,//picker-view第3列使用的数组
        year: date.getFullYear(),//当前选中的年
        month: 1,//当前选中的月
        day: 1,//当前选中的日
        value: [years.length - 1, 1, 1],
      },
      bindChange: function (e) {

        // picker-view当前选中的value，格式：{n,n,n}
        const val = e.detail.value

        //当前选中的年
        var yearTemp = this.data.years[val[0]]
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        //当前选中的月
        var monthTemp = this.data.months[val[1]]

        //当前选中的日
        var dayTemp = this.data.days[val[2]]

        //设置data，刷新页面
        this.setData({
          year: yearTemp,
          month: monthTemp,
          day: dayTemp
        })

        //打印年月日
        console.log(this.data.year + "年" + this.data.month + "月" +
this.data.day + "日")

        //定义一个数组，重新计算天数时使用
        const daysTemp = []

        //根据月份判断有多少天，31天
        if ((monthTemp == 1) || (monthTemp == 3) || (monthTemp == 5) ||
(monthTemp == 7) || (monthTemp == 8) || (monthTemp == 10) || (monthTemp == 12)) {
          for (let i = 1; i <= 31; i++) {
            daysTemp.push(i)
          }
        }
        //根据月份判断有多少天，30天
        else if ((monthTemp == 4) || (monthTemp == 6) || (monthTemp == 9) ||
(monthTemp == 11)) {
          for (let i = 1; i <= 30; i++) {
            daysTemp.push(i)
          }
        }
        //根据月份判断有多少天，闰年2月29天
        else if ((yearTemp % 4 == 0) || (yearTemp % 100 == 0) || (yearTemp %
400 == 0)) {
          for (let i = 1; i <= 29; i++) {
            daysTemp.push(i)
          }
        }
        // 根据月份判断有多少天，2月28天
        else {
          for (let i = 1; i <= 28; i++) {
            daysTemp.push(i)
          }
        }

        //设置data中days数组，刷新页面
        this.setData({
          days: daysTemp,
        })
      }
    })
```

非卖品，仅供非商业用途或交流学习使用

3.4.8 radio

radio-group 单项选择器，内部由多个 radio 组成。radio 组件为单选组件与 radio-group 组合使用，使用方式和 checkbox 相似。

radio-group 属性如表 3-28 所示。

表 3-28 radio-group 属性

属性	类型	默认值	说明
bindchange	EventHandle		<radio-group/> 中的选中项发生变化时触发 change 事件，event.detail = {value: 选中项 radio 的 value}

bindchange 绑定事件，在任何一个 radio 值改变的时候，都会触发该事件。返回对象通过 event.detail.value 获取所有 radio 的 name。

radio 属性如表 3-29 所示。

表 3-29 radio 属性

属性	类型	默认值	说明
value	String		<radio/> 标识。当该<radio/> 选中时，<radio-group/>的 change 事件会携带<radio/>的 value
checked	Boolean	false	当前是否选中
disabled	Boolean	false	是否禁用
color	Color		radio 的颜色，同 CSS 的 color

使用代码如下所示，展示效果如图 3-38 所示。

```
<!--pages/viewPage8/viewPage8.wxml-->
<!--绑定事件，当单击 radio 时调用-->
<radio-group bindchange="radioChange">
<!--label 常与 radio 和 checkbox 结合使用-->
    <label style="display: flex" wx:for-items="{{items}}">
        <radio  color="#ff5400"  value="{{item.name}}"  checked="{{item.
checked}}"/>{{item.value}}
    </label>
</radio-group>

// pages/viewPage8/viewPage8.js
Page({
  data: {
    items: [
      {name: 'CHN', value: '中国', checked: 'true'},
      {name: 'USA', value: '美国'},
      {name: 'BRA', value: '巴西'},
      {name: 'JPN', value: '日本'},
      {name: 'GBR', value: '英国'},
      {name: 'FRA', value: '法国'},
```

非卖品，仅供非商业用途或交流学习使用

```
      ]
    },
    radioChange: function(e) {
      console.log('radio 发生 change 事件，携带 value 值为: ', e)
      console.log('radio 发生 change 事件，携带 value 值为: ', e.detail.value)
    }
  })
```

如图 3-38 所示，小程序自带的 radio 是不能调整大小的，在项目中使用时很不方便，时常会影响整个界面的效果。为了解决这个问题，我使用 text 标签结合 icon 标签实现了 radio 效果。

这里我们实现一个选择地区的单选框。效果如图 3-39 所示。

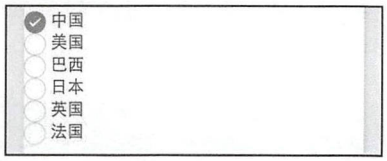


图 3-38 radio

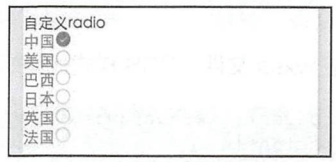


图 3-39 自定义 radio

图 3-39 的 icon 可以调整大小和位置。我们通过左边是 text，右边是 icon 来实现 radio 效果。

以列表方式排列所有地区，并给地区设置 isSelect 属性，如果 isSelect=true，则显示的 icon 的 type 为 success，否则 icon 的 type 显示为 circle。具体代码如下所示：

```
    <view class="section">自定义 radio</view>
    <view class="radio-group" >
          <label wx:for="{{areas}}" wx:for-item="area">
             <text bindtap="selectAreaOk" data-areaId="{{area.id}}">{{area.
value}} </text>
             <icon wx:if="{{area.isSelect}}" type="success" size="20"/>
             <icon wx:else type="circle" size="20"/>
          </label>
    </view>
```

上面代码遍历了 areas 数组用来显示地区名称和 icon，并为地区名称 text 设置了 bindtap 绑定单击事件用来跟 js 进行数据交互，data-areaId 是事件传递的参数。当 text 被单击时，代码根据地区的 id 来确定被单击的 text，被单击的 text 对应地区的 isSelect 属性设置为 true，否则设置为 false。然后，代码根据 area 对象的 isSelect 属性来确定显示的 icon，其中一个是圆圈，另一个是绿色的对钩。icon 的大小设置为 20，这里可以随意改变其大小。具体代码如下所示：

```
  //选择区域
    selectAreaOk: function (event) {

      //获取单击对象的 id
      var selectAreaId = event.target.dataset.areaid;
```

非卖品，仅供非商业用途或交流学习使用

```
    console.log('当前选中对象的 id 为: ', selectAreaId)

    //循环，判断对象是否选中
    for (var i = 0; i < this.data.areas.length; i++) {
      if (this.data.areas[i].id == selectAreaId) {
        this.data.areas[i].isSelect = true
      } else {
        this.data.areas[i].isSelect = false
      }
    }

    //重新赋值数组
    this.setData({
      areas: this.data.areas
    })
  }
```

对应 wxss 文件，CSS 样式代码如下所示：

```
/* pages/viewPage8/viewPage8.wxss */
.section {
  margin-top: 50px;
}
.radio-group{
    font-size: 20px;
    display: flex;
    flex-direction: column;
    color: #ff5400;
  }
```

3.4.9 slider

滑动选择器一般用于音乐播放、视频播放的进度调节。其属性如表 3-30 所示。

表 3-30　　slider 属性

属性	类型	默认值	说明
min	Number	0	最小值
max	Number	100	最大值
step	Number	1	步长，取值必须大于 0，并且可被(max 和 min)整除
disabled	Boolean	false	是否禁用
value	Number	0	当前取值
color	Color	#e9e9e9	背景条的颜色
selected-color	Color	#1aad19	已选择的颜色
show-value	Boolean	false	是否显示当前 value
bindchange	EventHandle		完成一次拖动后触发的事件，event.detail = {value: value}

非卖品，仅供非商业用途或交流学习使用

min 和 max 属性：分别设置滑块的取值范围，默认是 0～100。

step：滑块调节的步长，必须大于 0，并且是在 min 和 max 值之间的整数。

disabled：禁用组件。

value：当前滑块的值。

color：滑块底部背景条的颜色。

selected-color：滑块滑动区域的颜色。

show-value：是否在滑块最后显示滑块的值。

bindchange：绑定时间，每次滑块调节后会触发该事件。

滑动选择器，具体使用代码如下所示，显示效果如图 3-40 所示。

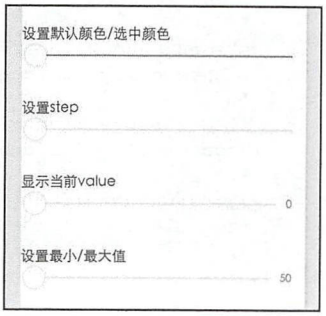


图 3-40 slider 控件使用

```
<!--pages/viewPage9/viewPage9.wxml-->
<view class="section">设置默认颜色/选中颜色
  <slider bindchange="sliderChange1" color="#ff5400" selected-color="#00a189"/>
</view>

<view class="section">设置 step
   <slider bindchange="sliderChange2" step="5"/>
</view>

<view class="section">显示当前 value
   <slider bindchange="sliderChange3" show-value/>
</view>

<view class="section">设置最小值/最大值
   <slider bindchange="sliderChange4" min="50" max="200" show-value/>
</view>

/* pages/viewPage9/viewPage9.wxss */
.section {
  margin-top: 50px;
}

// pages/viewPage9/viewPage9.js
Page({
  data:{},
  onLoad:function(options){
    // 页面初始化，options 为页面跳转所带来的参数
  },
  onReady:function(){
    // 页面渲染完成
  },
  onShow:function(){
    // 页面显示
  },
  onHide:function(){
    // 页面隐藏
  },
  onUnload:function(){
```

非卖品，仅供非商业用途或交流学习使用

```
    // 页面关闭
  },
  sliderChange1:function(e)
  {console.log('slider1 发生 change 事件，携带值为', e.detail.value)
  },
  sliderChange2:function(e)
  {console.log('slider2 发生 change 事件，携带值为', e.detail.value)
  },
  sliderChange3:function(e)
  {console.log('slider3 发生 change 事件，携带值为', e.detail.value)
  },
  sliderChange4:function(e)
  {console.log('slider4 发生 change 事件，携带值为', e.detail.value)
  },
})
```

3.4.10 switch

开关选择器，其属性如表 3-31 所示。

表 3-31　　switch 属性

属性	类型	默认值	说明
checked	Boolean	false	是否选中
type	String	switch	样式，有效值：switch、checkbox
bindchange	EventHandle		checked 改变时触发 change 事件，event.detail={value:checked}
color	Color		switch 的颜色，同 CSS 的 color

开关选择器使用代码如下所示，显示效果如图 3-41 所示。

```
<!--pages/viewPage10/viewPage10.wxml-->
<view class="section">
    <switch checked bindchange="switchChange1"/>
    <switch bindchange="switchChange2"/>
</view>


<!--switch 类型开关-->
<view class="section">switch 类型开关</view>
<switch type="switch" checked="true" bindchange="listenerSwitch"/>

<!--checkbox 类型开关-->
<view class="section">checkbox 类型开关</view>
<switch type="checkbox" bindchange="listenerCheckboxSwitch" />

/* pages/viewPage10/viewPage10.wxss */
.section {
  margin-top: 50px;
}
```

非卖品，仅供非商业用途或交流学习使用

```
// pages/viewPage10/viewPage10.js
Page({
   data: {
     switchSelect: true
   },

  switchChange1: function (e) {
    console.log('switch1 发生 change 事件，携带值为', e.detail.value)
  },
  switchChange2: function (e) {
    console.log('switch2 发生 change 事件，携带值为', e.detail.value)
  },

  //  switch 类型开关监听
   listenerSwitch: function (e) {
     console.log('switch 类型开关当前状态-----', e.detail.value);

   },

  // checkbox 类型开关监听
   listenerCheckboxSwitch: function (e) {
     console.log('checkbox 类型开关当前状态-----', e.detail.value)
   }
})
```

按钮和开关组件联合使用，代码如下所示，显示效果如图 3-42 所示。

```
<view class="section">按钮和开关组件联合使用</view>
<switch checked="{{switchSelect}}"/>
<button type="default" bindtap="openClick">打开</button>
<button type="default" bindtap="closeClick">关闭</button>

// 打开开关
  openClick: function () {
    this.setData({
      switchSelect: true
    })
  },

  // 关闭开关
  closeClick: function () {
    this.setData({
      switchSelect: false
    })
  }
```

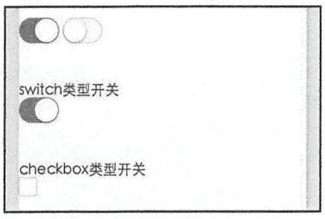


图 3-41 switch 控件使用

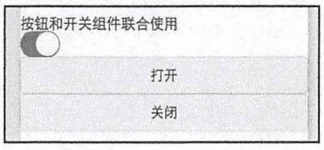


图 3-42 switch 和 button 结合使用

非卖品，仅供非商业用途或交流学习使用

使用 slider 和 switch 开关组件，设置一些数据值，代码如下所示，显示效果如图 3-43 所示。

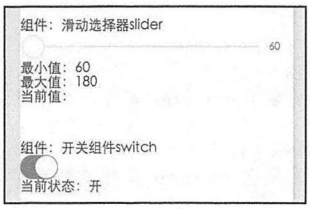


图 3-43 slider 和 switch 开关组件

```
    <view class="section">组件：滑动选择器 slider</view>
     <slider bindchange="sliderBindchange" min="{{min}}" max="{{max}}" show-
value/>
     <view>最小值：{{min}}</view>
     <view>最大值：{{max}}</view>
     <view>当前值：{{text}}</view>
     <view class="section"></view>
     <view>组件：开关组件 switch</view>
     <switch checked type="switch" bindchange="switchBindchange"/>
     <view>当前状态：{{switchState}}</view>

    // pages/viewPage10/viewPage10.js
    Page({
      data: {
        switchSelect: true,

        min:'60',
        max:'180',
        text:'',
        switchState:'开'
      },


      //滑块事件
      sliderBindchange:function(e){
        this.setData({
          text:e.detail.value
        })
      },

      //开关事件
      switchBindchange:function(e){
        if(e.detail.value){
          this.setData({
            switchState:'开'
          })
        }else{
          this.setData({
            switchState:'关'
          })
        }
```

```
    }
})
```

3.4.11 textarea

多行输入框，textarea 组件是由客户端创建的原生组件，它的层级是最高的。textarea 组件在 scroll-view 中使用。CSS 动画对 textarea 组件无效。

其属性如表 3-32 所示。

表 3-32 textarea 属性

属性	类型	默认值	说明
value	String		输入框的内容
placeholder	String		输入框为空时占位符
placeholder-style	String		指定 placeholder 的样式
placeholder-class	String	textarea-placeholder	指定 placeholder 的样式类
disabled	Boolean	false	是否禁用
maxlength	Number	140	最大输入长度，设置为-1 的时候不限制最大长度
auto-focus	Boolean	false	自动聚焦，拉起键盘
focus	Boolean	false	获取焦点
auto-height	Boolean	false	是否自动增高，设置 auto-height 时，style.height 不生效
fixed	Boolean	false	如果 textarea 是在一个 position:fixed 的区域，需要显示指定属性 fixed 为 true
cursor-spacing	Number	0	指定光标与键盘的距离，单位为 px。取 textarea 距离底部的距离和 cursor-spacing 指定距离的最小值作为光标与键盘的距离
bindfocus	EventHandle		输入框聚焦时触发，event.detail = {value: value}
bindblur	EventHandle		输入框失去焦点时触发，event.detail = {value: value}
bindlinechange	EventHandle		输入框行数变化时调用，event.detail = {height: 0, heightRpx: 0, lineCount: 0}
bindinput	EventHandle		当键盘输入时，触发 input 事件，event.detail = {value: value}，bindinput 处理函数的返回值并不会反映到 textarea 上

上面属性的具体使用，参考下面代码，显示效果如图 3-44 所示。

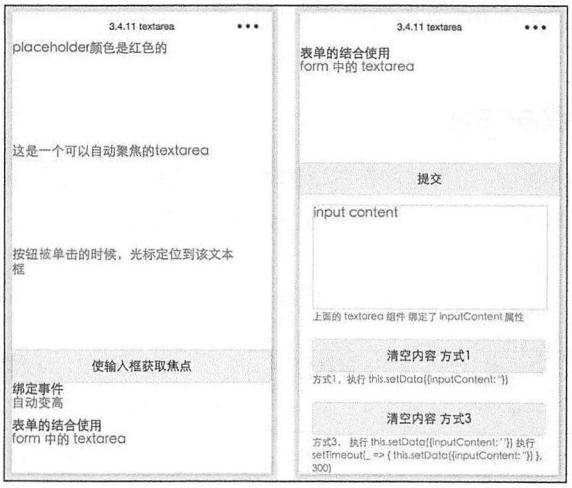


图 3-44　textarea 使用

```
    <!--pages/viewPage11/viewPage11.wxml-->

    <!--设置占位符，指定 placeholder 的样式-->
    <view>
      <textarea placeholder="placeholder 颜色是红色的" placeholder-style="color:red;" />
    </view>

    <!--初始化光标定位到该文本框-->
    <view>
      <textarea placeholder="这是一个可以自动聚焦的 textarea" auto-focus />
    </view>

    <!--单击按钮，定位光标-->
    <view>
      <textarea placeholder="按钮被单击的时候，光标定位到该文本框" focus="{{focus}}" />
        <button bindtap="bindButtonTap">使输入框获取焦点</button>
      </view>

      <!--bindfocus:输入框聚焦时触发
    bindblur:输入框失去焦点时触发
    bindlinechange:输入框行数变化时调用
    bindinput:当键盘输入时，触发 input 事件-->
      <view>绑定事件
        <textarea bindfocus="bindfocusFun" bindblur="bindblurFun" bindlinechange=
"bindlinechangeFun" bindinput="bindinputFun" auto-height placeholder="自动变高" />
      </view>
      <view class="section">表单的结合使用
        <form bindsubmit="bindFormSubmit">
          <textarea placeholder="form 中的 textarea" name="textarea" />
          <button form-type="submit"> 提交 </button>
        </form>
      </view>


      <!--使用 view 设置带边框的文本框-->
```

```
    <view style="width: 90%; margin: 20rpx auto 0;">
      <textarea placeholder="input content" value="{{inputContent}}" style=
"margin-top: 30rpx; border: 1rpx solid #ddd; width: 100%;" />
      <text style="color: #999; font-size: 28rpx;">上面的 textarea 组件绑定了
inputContent 属性</text>
    </view>

  /* pages/viewPage11/viewPage11.wxss */
  .section {
    margin-top: 10px;
  }

  // pages/viewPage11/viewPage11.js
  Page({
    data: {
      height: 20,
      focus: false,
      inputContent: ''
    },
    // bindfocus:输入框聚焦时触发
    bindfocusFun: function (e) {
      console.log(e.detail.value)
    },
    // bindblur:输入框失去焦点时触发
    bindblurFun: function (e) {
      console.log(e.detail.value)
    },
    // bindlinechange:输入框行数变化时调用
    bindlinechangeFun: function (e) {
      console.log(e.detail.value)
    },
    // bindinput:当键盘输入时，触发 input 事件
    bindinputFun: function (e) {
      console.log(e.detail.value)
    },


    // 按钮单击事件
    bindButtonTap: function () {
      this.setData({
        focus: true
      })
    },

    // 表单提交事件
    bindFormSubmit: function (e) {
      console.log(e.detail.value.textarea)
    }
  })
```

微信小程序中 textarea 没有 bindchange 事件，所以无法在输入时给变量赋值。

虽然可以使用 bindblur 事件，但是绑定 bindblur 事件后，如果再单击按钮，则先执行完按钮事件后，再去执行 bindblur 事件，所以在 js 文件取不到输入值。解决方法：结合 from 表单，textarea 文本框输入后，再去单击提交按钮，这时会先执行 textarea 事件（获取文本

框输入内容），再去执行数据提交。

2 种清空文本框的方式，使用代码如下所示：

```
    <!--2 种清空文本框的方式-->
      <view style="width: 90%; margin: 50rpx auto 0;">
        <button bindtap="clearInputContent" data-mode="1">清空内容 方式 1</button>
        <text style="color: #999; font-size: 28rpx;">方式 1，执行 this.setData
({inputContent: ''})</text>
      </view>

      <view style="width: 90%; margin: 50rpx auto 0;">
        <button bindtap="clearInputContent" data-mode="2">清空内容 方式 3</button>
        <text style="color: #999; font-size: 28rpx;">方式 3，执行 this.setData
({inputContent: ' '}) 执行 setTimeout(_ => { this.setData({inputContent:
''}) }, 300)
        </text>
      </view>

    // 清空文本框
      clearInputContent(e) {
        const mode = parseInt(e.target.dataset.mode)

        switch (mode) {
          case 1:
            this.setData({
              inputContent: ''
            })
            break;
          case 2:
            this.setData({
              inputContent: ' '
            })
            setTimeout(_ => {
              this.setData({
                inputContent: ''
              })
            }, 300)
            break;
        }
      }
```

3.5 操作反馈

表单组件主要有：action-sheet、modal、toast、loading。但是这些组件都可以使用 API 实现，本节只做简单介绍。

3.5.1 action-sheet

action-sheet 组件可以使用 API:wx.showActionSheet 实现类似效果。

action-sheet 组件是从底部弹出的选择菜单。action-sheet 有两个子组件，action-

sheet-item 为子选项，action-sheet-cancel 为取消选项，与 action-sheet-item 中间有间隔。开发者可以通过为 action-sheet-item 添加 bindtap 来触发单击后作出的响应。action-sheet-cancel 被单击时会触发 action-sheet 的 bindchange 事件。在 bindchange 绑定的函数中可以控制菜单的显示。另外，单击空白处时菜单也会隐藏。

action-sheet（上拉菜单）是从屏幕底部出现的菜单，其属性如表 3-33 所示。

表 3-33　　action-sheet 属性

属性	类型	默认值	说明
hidden	Boolean	true	是否隐藏
bindchange	EventHandle		单击背景或 action-sheet-cancel 按钮时触发 change 事件，不携带数据

action-sheet-item：底部菜单表的子选项。

action-sheet-cancel：底部菜单表的取消按钮，和 action-sheet-item 的区别是，单击它会触发 action-sheet 的 change 事件，并且外观上会同它上面的内容间隔开来。

具体使用代码如下所示，显示效果如图 3-45 所示。

```
    <!--pages/viewPage1/viewPage1.wxml-->
    <button type="default" bindtap="actionSheetTap">弹出 action sheet</button>
    <action-sheet hidden="{{actionSheetHidden}}" bindchange="actionSheetChange">
        <block wx:for-items="{{actionSheetItems}}">
            <action-sheet-item class="item" bindtap="clickFrom{{item}}"> {{item}}
</action-sheet-item>
        </block>
        <action-sheet-cancel class="cancel">取消</action-sheet-cancel>
    </action-sheet>

    // pages/viewPage1/viewPage1.js

    var items = ['item1', 'item2', 'item3', 'item4']
    Page({
      data:{
        actionSheetHidden: true,
        actionSheetItems: items
      },
      //显示弹框
      actionSheetTap: function(e) {
        this.setData({
          actionSheetHidden: !this.data.actionSheetHidden
        })
      },
      clickFromitem1:function()
      {
        console.log("item1 click")
      },
      clickFromitem2:function()
      {
        console.log("item2 click")
      },
      clickFromitem3:function()
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
  {
    console.log("item3 click")
  },
  clickFromitem4:function()
  {
    console.log("item4 click")
  },
  //单击背景或取消按钮
  actionSheetChange: function(e) {
    this.setData({
      actionSheetHidden: !this.data.actionSheetHidden
    })
  }
})
```

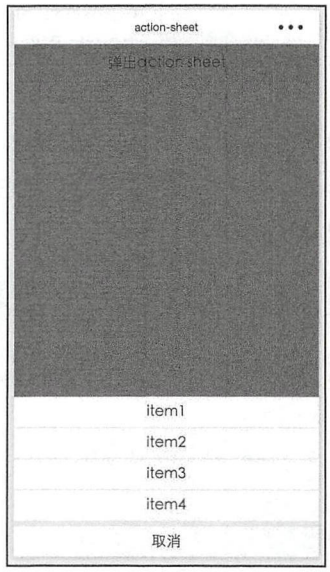


图 3-45 action-sheet 使用

可以使用 wx.showActionSheet 弹出自定义操作菜单，代码如下所示：

```
<!--pages/viewPage1/viewPage1.wxml-->
<view class="section">使用 wx.showActionSheet 显示操作菜单</view>
<button type="default" bindtap="buttonClick">弹出 action sheet</button>

// pages/viewPage1/viewPage1.js
// 使用 wx.showActionSheet 显示操作菜单
  buttonClick: function() {
    wx.showActionSheet({
      itemList:['item1','item2','item3','item4'],success:function(res)
      {
        if(!res.cancel)
        {
          console.log(res.tapIndex);
        }
      }
    });
  }
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

上面代码定义了 4 个菜单，分别是 item1、item2、item3、item4。单击之后，调用 success:function，代码通过 res 来判断是否单击了取消（cancel）按钮，如果不是单击取消按钮，可以通过 res.tapIndex 来判断单击的是哪一个菜单。

3.5.2 modal

modal 组件可以使用 API: wx.showModal 实现类似效果。

modal 类似于 JavaScript 中的 confirm 弹框，弹框常用于提示一些信息，比如：退出应用、清空缓存、发布帖子提交时一些提示等。modal 默认情况下是一个带有确认取消的弹框，不过单击取消后弹框不会自动隐藏，需要通过触发事件调用函数来控制 hidden 属性。

modal（对话弹框）属性如表 3-34 所示。

表 3-34 modal 属性

属性	类型	默认值	说明
title	String		标题
hidden	Boolean	false	是否隐藏整个弹框
no-cancel	Boolean	false	是否隐藏 cancel 按钮
confirm-text	String	确定	confirm 按钮文字
cancel-text	String	取消	cancel 按钮文字
bindconfirm	EventHandle		单击确认触发的回调
bindcancel	EventHandle		单击取消以及蒙层触发的回调

具体使用代码如下所示，显示效果如图 3-46 所示。

```
    <!--pages/viewPage2/viewPage2.wxml-->
    <modal title="标题" confirm-text="确定" cancel-text="取消" hidden=
"{{modalHidden}}" bindconfirm="modalChange1" bindcancel="modalChange2">
```

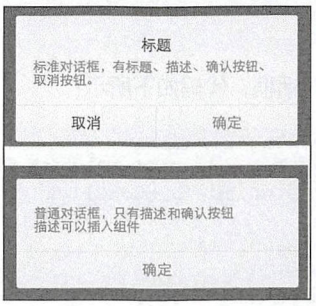


图 3-46 modal 使用

标准对话框，有标题、描述、确认按钮、取消按钮。普通对话框代码如下所示：

```
    </modal>

    <modal class="modal" hidden="{{modalHidden2}}" no-cancel bindconfirm=
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
"modalChange2">
        <view> 普通对话框，只有描述和确认按钮 </view>
        <view> 描述可以插入组件 </view>
    </modal>

    <view class="btn-area">
        <button type="default" bindtap="modalTap1">单击弹出 modal1</button>
        <button type="default" bindtap="modalTap2">单击弹出 modal2</button>
    </view>

    // pages/viewPage2/viewPage2.js
    Page({
      data: {
        modalHidden: true,
        modalHidden2: true
      },
      //按钮事件 1
      modalTap1: function(e) {
        this.setData({
          modalHidden: false
        })
      },
      //按钮事件 2
      modalTap2: function(e) {
        this.setData({
          modalHidden2: false
        })
      },
      //弹框按钮 1
      modalChange1: function(e) {
        this.setData({
          modalHidden: true
        })
      },
      // 弹框按钮 2
      modalChange2: function(e) {
        this.setData({
          modalHidden2: true
        })
      },
    })
```

使用 wx.showModal 显示对话框，代码如下所示：

```
    <!--pages/viewPage2/viewPage2.wxml-->
    <view class="section">使用 wx.showModal 显示对话框</view>
    <button type="default" bindtap="buttonClick">弹出 modal</button>

    // pages/viewPage2/viewPage2.js
    //使用 wx.showModal 显示对话框
      buttonClick: function () {
        wx.showModal({
          title: '标题',
          content: '这里是描述', success: function (res) {
            if (res.confirm) {
              console.log("确定按钮");
            }
            else {
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
            console.log("取消按钮");
          }
        }
      });
    }
```

上面代码设置了 title 和 content，并通过 success:function 中的 res.confirm 来判断单击的是确认按钮还是取消按钮。

3.5.3 toast

toast 组件可以使用 API:wx.showToast 实现类似效果，取消使用 wx.hideToast。

toast（消息提示框）可用于提示一些信息，比如清除缓存后给出一个友好的提示！toast 也可以做到遮蔽其他操作，因为 toast 显示时其他操作是无效的，相当于遮罩层。

toast（消息提示框）属性如表 3-35 所示。

表 3-35 toast 属性

属性	类型	默认值	说明
duration	Float	1500	hidden 设置为 false 后，触发 bindchange 的延时，单位为 ms
hidden	Boolean	false	是否隐藏
bindchange	EventHandle		duration 延时后触发

使用代码如下所示，显示效果如图 3-47 所示。

图 3-47 toast 使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
<!--pages/viewPage3/viewPage3.wxml-->
<view class="body-view">
    <toast hidden="{{toastHidden1}}" bindchange="toastChange1">
        默认
    </toast>
    <button type="default" bindtap="toastTap1" >单击弹出默认 toast</button>
</view>
<view class="body-view">
    <toast hidden="{{toastHidden2}}" duration="3000" bindchange="toastChange2">
        设置 duration
    </toast>
    <button type="default" bindtap="toastTap2">单击弹出设置 duration 的 toast
</button>
</view>

// pages/viewPage3/viewPage3.js
Page({
  data:{
    toastHidden1:true,
    toastHidden2:true
  },

  // 按钮 1 事件
  toastTap1:function(){
    this.setData({
      toastHidden1:false
    })
  },
  //按钮 2 事件
  toastTap2:function(){
    this.setData({
    toastHidden2:false
    })
  },
  //默认 1500ms，时间到调用该事件
  toastChange1:function(){
    console.log("toastChange1")
    this.setData({
    toastHidden1:true
    })
  },
  //设定的 duration=3000ms，时间到调用该事件
  toastChange2:function(){
    console.log("toastChange2")
  }
})
```

使用 wx.showToast 显示提示框，代码如下所示：

```
<!--pages/viewPage3/viewPage3.wxml-->
<view class="section">使用 wx.showToast 显示提示框</view>
<button type="default" bindtap=" buttonClick">弹出 Toast</button>

// pages/viewPage3/viewPage3.js
  //使用 wx.showToast 显示对话框
  buttonClick: function () {
    wx.showToast({
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        title: '成功',
        icon: 'success',
        duration: 2000
      });

      //延时隐藏提示框
      setTimeout(function () {
        wx.hideToast()
      }, 1000);
    }
```

3.5.4 loading

loading 组件可以使用 API:wx.showToast 实现类似效果，是请求网络数据时的一种方式，通过 hidden 属性设置显示与否。

loading（加载提示框）属性如表 3-36 所示。

表 3-36 loading 属性

属性	类型	默认值	说明
hidden	Boolean	false	是否隐藏

使用代码如下所示，显示效果如图 3-48 所示。

```
<!--pages/viewPage4/viewPage4.wxml-->
<view class="body-view">
    <loading hidden="{{hidden}}">
        加载中...
    </loading>
    <button type="default" bindtap="loadingTap">单击显示 loading</button>
</view>

// pages/viewPage4/viewPage4.js
Page({
    data: {
        hidden: true
    },

    //弹出 loading 按钮事件
    loadingTap: function(){

        var that = this
        that.setData({
          hidden: false
        });
        //设置 3000ms 之后执行
        setTimeout(function(){
          that.setData({
              hidden: true
          });
          that.update();
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        }, 3000);
    }
})
```

图 3-48 loading 使用

可以使用 wx.ShowNavigationBarLoading，在标题栏显示加载，代码如下所示：

```
<!--pages/viewPage4/viewPage4.wxml-->
<view class="section">使用 wx.ShowNavigationBarLoading 显示加载</view>
<button type="default" bindtap="buttonClick">显示</button>

// pages/viewPage4/viewPage4.js
//使用 wx.ShowNavigationBarLoading 显示加载
  buttonClick: function () {

    wx.showNavigationBarLoading()

    //延时隐藏提示框
    setTimeout(function () {
      wx.hideNavigationBarLoading()
    }, 1000);
  }
```

3.6 导航（navigator）

navigator 跳转分为两个状态，一种是关闭当前页面，另一种是不关闭当前页面。navigator 用 redirect 属性指定。

代码如下所示：

```
<navigator url="../pageView1/pageView1">单击跳转不关闭当前页面</navigator>
<navigator url="../pageView2/pageView2" redirect="true" >单击跳转关闭当前页面</navigator>
```

navigator 页面链接，属性如表 3-37 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 3-37 navigator 属性

属性	类型	默认值	说明
target	String		在哪个目标上发生跳转，默认为当前小程序
url	String		当前小程序内的跳转链接
open-type	String	navigate	可选值: 'navigate'：对应 wx.navigateTo 或 wx.navigate ToMiniProgram 的功能 'redirect': 对应 wx.navigateTo 的功能 'switchTab': 对应 wx.switchTab 的功能 'reLaunch': 对应 wx.reLaunch 的功能 'navigateBack'：对应 wx.navigateBack 的功能 'exit': 退出小程序，target="miniProgram" 时生效
delta	Number		当 open-type 为 'navigateBack' 时有效，表示回退的层数
app-id	String		当 target="miniProgram"时有效，表示要打开的小程序 appId
path	String		当 target="miniProgram"时有效，表示打开的页面路径，如果为空则打开首页
extra-data	Object		当 target="miniProgram"时有效，表示需要传递给目标小程序的数据，目标小程序可在 App.onLaunch()、App.onShow()中获取到这份数据
version	version	release	当 target="miniProgram"时有效，表示要打开的小程序版本，有效值 develop（开发版）、trial（体验版）、release（正式版），仅在当前小程序为开发版或体验版时此参数有效；如果当前小程序是正式版，则打开的小程序必须是正式版
hover-class	String	navigator-hover	指定单击时的样式类，当 hover-class="none"时，没有单击态效果
hover-stop-propagation	Boolean	false	指定是否阻止本节点的组件节点出现单击态
hover-start-time	Number	50	按住后多久出现单击态，单位为 ms
hover-stay-time	Number	600	手指松开后单击态保留时间，单位为 ms

redirect 将被废弃，开发者可以使用 open-type 属性替代，具体代码如下所示：

```
    <!--使用 open-type 属性-->
    <navigator  url="../pageView1/pageView1"  open-type="navigate">open-type=
"navigate" </navigator>
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    <navigator url="../pageView1/pageView1" open-type="redirect">open-type=
"redirect" </navigator>
    <navigator url="../pageView1/pageView1" open-type="switchTab">open-type=
"switchTab" </navigator>
```

hover-class 指定单击时的样式类，默认为 {background-color: rgba(0, 0, 0, 0.1); opacity: 0.7;}, <navigator/> 的子节点背景色应为透明色。具体使用代码如下所示：

```
    <view class="btn-area">
      <navigator url="../pageView1/pageView1?title=navigate" hover-class=
"navigator- hover">跳转到新页面</navigator>
      <navigator url="redirect?title=redirect" open-type="redirect" hover-
class="other-navigator-hover">在当前页打开</navigator>
      <navigator url="index" open-type="switchTab" hover-class="other-
navigator-hover">切换 Tab</navigator>
    </view>
```

navigator 跳转 url 传参，也就是问号传值，跳到新页面之后，在 onload 里面直接接收参数，接收方法也就是 options.参数名。具体代码如下所示：

```
    <!--index.wxml-->
    <view class="btn-area">
      <navigator url="../pageView1/pageView1?title=navigate" hover-class=
"navigator-hover">跳转到新页面</navigator>
      <navigator url="redirect?title=redirect" open-type="redirect" hover-
class="other-navigator-hover">在当前页打开</navigator>
      <navigator url="index" open-type="switchTab" hover-class="other-
navigator-hover">切换 Tab</navigator>
    </view>

    <!--pages/pageView1/pageView1.wxml-->
    <view> 收到的参数 title 是: </view>
    <view style="text-align:center"> {{title}} </view>

    // pages/pageView1/pageView1.js
    Page({
      data:{
        title:"新页面"
      },
      onLoad: function(options) {
        this.setData({
          title: options.title
        })
      }
    })
```

可以使用 wx.navigateTo 和 wx.redirectTo 推出新页面，代码如下所示：

```
    <!--index.wxml-->
    <button type="default" bindtap="buttonClick">使用 wx.navigateTo</button>
    <button type="default" bindtap="buttonClick2">使用 wx.redirectTo</button>

    //index.js
    Page({
      data: {
      },
      // 使用 wx.navigateTo
      buttonClick: function () {
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    wx.navigateTo({ url: '../pageView1/pageView1' })
  },
  // 使用 wx.redirectTo
  buttonClick2: function () {
    wx.redirectTo({ url: '../pageView2/pageView2' })
  }
})
```

wx.navigateTo 推出的界面，可以使用 wx.navigateBack();返回上一界面。

小程序从基础库版本 2.1.0 开始支持 functional-page-navigator 组件。该组件仅在插件的自定义组件中有效，用于跳转到插件功能页。这个组件本身可以在开发者工具中使用，但功能页的跳转目前不支持在开发者工具中调试，请在真机上调试。功能页是插件所有者小程序中的一个特殊页面，开发者不能自定义这个页面的外观。在功能页展示时，一些与页面展示相关的接口将被禁用（接口调用返回 fail）。

functional-page-navigator 的属性如表 3-38 所示。

表 3-38 functional-page-navigator 的属性

属性名	类型	默认值	说明
version	String	release	跳转到的小程序版本，有效值 develop（开发版）、trial（体验版）、release（正式版）；**线上版本必须设置为 release **
name	String		要跳转到的功能页
args	Object	null	功能页参数，参数格式与具体功能页相关
bindsuccess	EventHandle		功能页返回，且操作成功时触发， detail 格式与具体功能页相关
bindfail	EventHandle		功能页返回，且操作失败时触发， detail 格式与具体功能页相关

name 的有效值如表 3-39 所示。

表 3-39 name 的有效值

值	说明	接收的 args	success 返回的 detail	fail 返回的 detail
loginAndGetUserInfo	获取用户信息，对应 wx.login 和 wx.getUserInfo	与 wx.getUserInfo 接收的 args 相同（除回调函数外）	wx.login 和 wx.getUserInfo 结果的并集	与 wx.login 或 wx.getUserInfo 相同
requestPayment	支付，对应 wx.requestPayment	fee 字段，表示需要显示在页面中的金额，单位为分；paymentArgs 字段可读取任意数据，并传递给功能页中的响应函数	与 wx.requestPayment 相同	与 wx.requestPayment 相同

非卖品，仅供非商业用途或交流学习使用

使用示例代码如下所示：

```
    < !--sample.wxml -- >
      <functional-page - navigator name = "loginAndGetUserInfo" bind: success =
"loginSuccess" >
      <button>登录到插件 < /button>
      < /functional-page-navigator>
```

```
    // redirect.js navigator.js
    Component({
       methods: {
         loginSuccess: function(e) {
        console.log(e.detail.code) // wx.login 的 code
          console.log(e.detail.userInfo) // wx.getUserInfo 的 userInfo
      }
      }
    })
```

3.7 媒体组件

媒体组件由音频组件 audio、图片组件 image、视频组件 video 组成。开发者可以使用媒体组件来丰富小程序的界面内容。

3.7.1 audio

audio 为音频组件，我们可以轻松地在小程序中播放音频。audio 组件属性如表 3-40 所示。

表 3-40　　audio 属性

属性	类型	默认值	说明
id	String		audio 组件的唯一标识符
src	String		要播放音频的资源地址
loop	Boolean	false	是否循环播放
controls	Boolean	true	是否显示默认控件
poster	String		默认控件上的音频封面的图片资源地址，如果 controls 属性值为 false，则设置 poster 无效
name	String	未知音频	默认控件上的音频名字，如果 controls 属性值为 false，则设置 name 无效
author	String	未知作者	默认控件上的作者名字，如果 controls 属性值为 false，则设置 author 无效
binderror	EventHandle		当发生错误时触发 error 事件，detail = {errMsg: MediaError.code}
bindplay	EventHandle		当开始/继续播放时触发 play 事件

非卖品，仅供非商业用途或交流学习使用

续表

属性	类型	默认值	说明
bindpause	EventHandle		当暂停播放时触发 pause 事件
bindtimeu-pdate	EventHandle		当播放进度改变时触发 timeupdate 事件，detail = {currentTime, duration}
bindended	EventHandle		当播放到末尾时触发 ended 事件

id：标注唯一组件以 this.audioCtx = wx.createAudioContext('myAudio')获取控制组件的对象。

src：音频地址。

loop：是否循环播放。

controls：是否显示默认控件，微信音乐显示的组件，如图 3-49 所示。

poster：音频封面图片，默认控件上的音频封面的图片资源地址，如果 controls 属性值为 false，则设置 poster 无效。

name：歌名，如果 controls 属性值为 false，则设置 name 无效。

author：歌手，如果 controls 属性值为 false，则设置 author 无效。

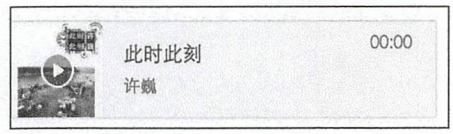


图 3-49 controls 样式

binderror：播放错误时调用，携带参数 detail = {errMsg: MediaError.code}，通过 e.detail 来获取错误信息{errMsg: MediaError.code}，MediaError.code 有 4 种错误值，如表 3-41 所示。

表 3-41 MediaError.code 值

返回错误码	描述
MEDIA_ERR_ABORTED	获取资源被用户禁止
MEDIA_ERR_NETWORD	网络错误
MEDIA_ERR_DECODE	解码错误
MEDIA_ERR_SRC_NOT_SUPPOERTED	不合适资源

bindplay：播放时触发该事件。

bindpause：停止时触发该事件。

bindtimeupdate：时间改变时触发，该函数携带有参数 detail={currentTime, duration}，即当前时间和持续播放时间。

bindended：播放结束时触发。

audio 使用代码如下所示。显示效果如图 3-50 所示。

```
    <!--pages/pageView1/pageView1.wxml-->
    <audio  poster="{{poster}}"  name="{{name}}"  author="{{author}}"  src=
"{{src}}" id="myAudio" controls loop></audio>
```

非卖品，仅供非商业用途或交流学习使用

```
<button type="primary" bindtap="audioPlay">播放</button>
<button type="primary" bindtap="audioPause">暂停</button>
<button type="primary" bindtap="audio14">设置当前播放时间为 14s</button>
<button type="primary" bindtap="audioStart">重新开始</button>

// pages/pageView1/pageView1.js
Page({
  onReady: function (e) {
    // 使用 wx.createAudioContext 获取 audio 上下文 context
    this.audioCtx = wx.createAudioContext('myAudio')
  },
  data: {
    poster: ' https://wx.leadingdo.com/audios/test.jpg',
    name: '此时此刻',
    author: '许巍',
    src: ' https://wx.leadingdo.com/audios/test.mp3',
  },

  // 播放
  audioPlay: function () {
    this.audioCtx.play()
  },
  // 暂停
  audioPause: function () {
    this.audioCtx.pause()
  },
  // 从 14s 开始播放
  audio14: function () {
    this.audioCtx.seek(14)
  },
  // 重新开始播放
  audioStart: function () {
    this.audioCtx.seek(0)
  }
})
```

图 3-50 audio

非卖品，仅供非商业用途或交流学习使用

3.7.2 image

image 也是一个程序不可或缺的图片组件，可以这样说，一个 App 中 image 组件随处可以看到。image 属性如表 3-42 所示。

表 3-42 image 属性

属性	类型	默认值	说明
src	String		图片资源地址
mode	String	'scaleToFill'	图片裁剪、缩放的模式
binderror	HandleEvent		当错误发生时，发布到 AppService 的事件名，事件对象 event.detail = {errMsg: 'something wrong'}
bindload	HandleEvent		当图片载入完毕时，发布到 AppService 的事件名，事件对象 event.detail = {height:'图片高度 px', width:'图片宽度 px'}

image 有两种加载方式，第 1 种是加载网络图片，第 2 种是加载本地图片，都用 src 属性去指定。代码如下所示：

```
    <!--pages/pageView2/pageView2.wxml-->

    <!--加载本地图片-->
    <image class="widget__arrow" src="../image/XiBao.jpg" mode="aspectFill">
    </image>

    <!--加载网络图片，绑定变量-->
    <image class="image_frame" src="{{imageUrl}}" mode="aspectFill">
    </image>

    <!--加载网络图片，直接指定 url-->
    <image      class="image_frame"      src="https://wx.leadingdo.com/audios/
JiTuanlogo.png " mode="aspectFill">
    </image>
```

binderror：绑定事件，发生错误时调用该方法。binderror 通过 event.detail 来获取错误信息{errMsg: 'something wrong'}。

bindload：绑定事件，图片载入完毕时调用该方法。bindload 可以通过 event.detail 来获取图片的高度和宽度{height:'图片高度 px', width:'图片宽度 px'}。

给 image 设置 bindload 属性，绑定一个函数，这个函数可以获取到原图的宽度和高度。代码如下所示：

```
    <!--pages/pageView2/pageView2.wxml-->
    <!--绑定事件-->
    <image  class="image_frame"  src="{{imageUrl}}"  mode="aspectFill"  bindload=
"imageLoad">
    </image>
```

非卖品，仅供非商业用途或交流学习使用

```
// pages/pageView2/pageView2.js
//bindload 绑定的事件
  imageLoad: function (e) {
    var width = e.detail.width    //获取图片真实宽度
    var height = e.detail.height

      console.log("图片宽度: " + width)

      console.log("图片高度: " + height)
  }
```

image 组件默认宽度为 300px、高度为 225px。显示图片时常会用到一些裁剪和缩放的属性 mode。mode 有 13 种模式，其中 4 种是缩放模式，9 种是裁剪模式。

缩放模式，如表 3-43 所示。

表 3-43　　缩放模式

模式	说明
scaleToFill	不保持纵横比缩放图片，使图片的宽高完全拉伸至填满 image 元素
aspectFit	保持纵横比缩放图片，使图片的长边能完全显示出来。也就是说，可以完整地将图片显示出来
aspectFill	保持纵横比缩放图片，只保证图片的短边能完全显示出来。也就是说，图片通常只在一个方向上是完整的，另一个方向将会发生截取
widthFix	宽度不变，高度自动变化，保持原图宽高比不变

裁剪模式，如表 3-44 所示。

表 3-44　　裁剪模式

模式	说明
top	不缩放图片，只显示图片的顶部区域
bottom	不缩放图片，只显示图片的底部区域
center	不缩放图片，只显示图片的中间区域
left	不缩放图片，只显示图片的左边区域
right	不缩放图片，只显示图片的右边区域
top left	不缩放图片，只显示图片的左上边区域
top right	不缩放图片，只显示图片的右上边区域
bottom left	不缩放图片，只显示图片的左下边区域
bottom right	不缩放图片，只显示图片的右下边区域

在 js 文件中，定义一个 array 数组，存储 4 种缩放模式参数，9 种裁剪模式参数，在 wxml 中通过 wx:for 循环，创建 image 组件。具体使用代码如下所示：

```
<!--pages/pageView2/pageView2.wxml-->
<view class="section" wx:for="{{array}}" wx:for-item="item">
  <view class="section__title">{{item.text}}</view>
  <image style="width: 300px; height: 200px; background-color: #eeeeee;"
```

非卖品，仅供非商业用途或交流学习使用

```
mode="{{item.mode}}" src="{{src}}"></image>
    </view>

    // pages/pageView2/pageView2.js
    Page({
      data: {
        imageUrl:  "http://www.wuxianedu.com/uploads/allimg/161116/1-161116I9220-
L.jpg",
        src: '../image/book.png',
        array: [
          {
            mode: 'scaleToFill',
            text: 'scaleToFill: 不保持纵横比缩放图片，使图片的宽高完全拉伸至填满'
          },
          {
            mode: 'aspectFit',
            text: 'aspectFit: 保持纵横比缩放图片，使图片的长边能完全显示出来'
          },
          {
            mode: 'aspectFill',
            text: 'aspectFill: 保持纵横比缩放图片，只保证图片的短边能完全显示出来'
          },
          {
            mode: 'widthFix',
            text: 'widthFix: 宽度不变，高度自动变化，保持原图宽高比不变'
          },
          {
            mode: 'top',
            text: 'top: 不缩放图片，只显示图片的顶部区域'
          }, {
            mode: 'bottom',
            text: 'bottom: 不缩放图片，只显示图片的底部区域'
          }, {
            mode: 'center',
            text: 'center: 不缩放图片，只显示图片的中间区域'
          }, {
            mode: 'left',
            text: 'left: 不缩放图片，只显示图片的左边区域'
          }, {
            mode: 'right',
            text: 'right: 不缩放图片，只显示图片的右边区域'
          }, {
            mode: 'top left',
            text: 'top left: 不缩放图片，只显示图片的左上边区域'
          }, {
            mode: 'top right',
            text: 'top right: 不缩放图片，只显示图片的右上边区域'
          }, {
            mode: 'bottom left',
            text: 'bottom left: 不缩放图片，只显示图片的左下边区域'
          }, {
            mode: 'bottom right',
            text: 'bottom right: 不缩放图片，只显示图片的右下边区域'
          }]
      },
      imageError: function (e) {
        console.log('image3 发生 error 事件，携带值为', e.detail.errMsg)
      },
```

```
    //bindload 绑定的事件
    imageLoad: function (e) {
      var width = e.detail.width    //获取图片真实宽度
      var height = e.detail.height

      console.log("图片宽度: " + width)

      console.log("图片高度: " + height)
    }
  })
```

图片原图如图 3-51 所示。

4 种缩放模式显示效果，如图 3-52 所示。

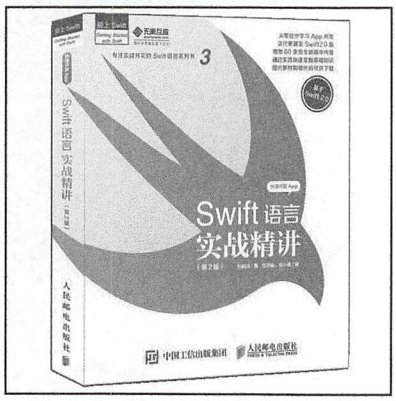


图 3-51　图片原图

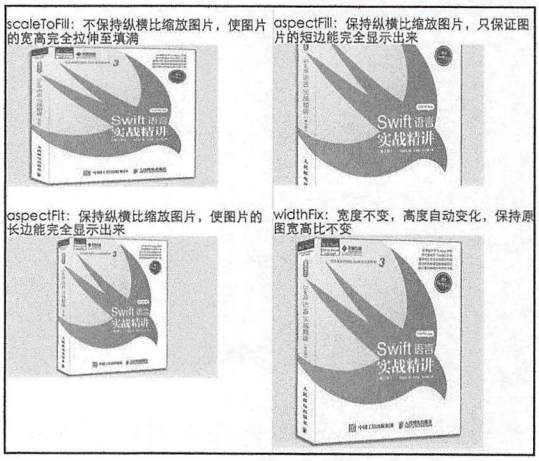


图 3-52　4 种缩放模式

9 种裁剪模式显示效果，如图 3-53 所示。

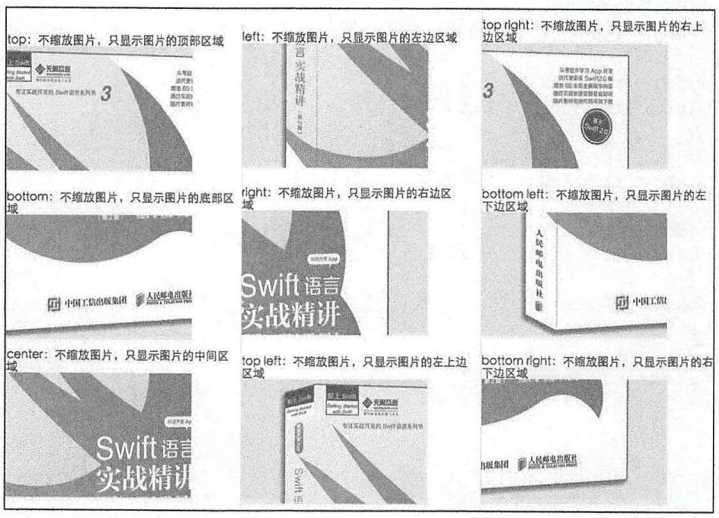


图 3-53　9 种裁剪模式

3.7.3 video

视频播放组件的使用与图片组件的使用类似。video 组件不能在 scroll-view 中使用，CSS 动画对 video 组件无效。

video 组件通过 src 传递视频地址来播放视频。video 标签默认宽度为 300px、高度为 225px。设置 video 标签的宽高需要开发者通过 wxss 设置 width 和 height。video 组件的引用格式如下所示。显示效果如图 3-54 所示。

```
<!--pages/pageView3/pageView3.wxml-->
<video    src="http://wx.leadingdo.com/videos/test.mp4"
binderror="videoErrorCallback"></video>

/* pages/pageView3/pageView3.wxss */
.videoC{
    width: 100%;
    height: 200px;
}
```

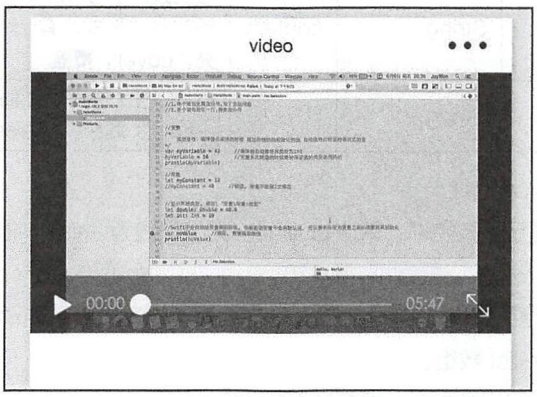


图 3-54 video 组件

video 属性如表 3-45 所示。

表 3-45 video 属性

属性	类型	默认值	说明
src	String		要播放视频的资源地址
duration	Number		指定视频时长
controls	Boolean	true	是否显示默认播放控件（播放/暂停按钮、播放进度、时间）
danmu-list	Object Array		弹幕列表
danmu-btn	Boolean	false	是否显示弹幕按钮，只在初始化时有效，不能动态变更

续表

属性	类型	默认值	说明
enable-danmu	Boolean	false	是否展示弹幕，只在初始化时有效，不能动态变更
autoplay	Boolean	false	是否自动播放
loop	Boolean	false	是否循环播放
muted	Boolean	false	是否静音播放
bindplay	EventHandle		当开始/继续播放时触发 play 事件
bindpause	EventHandle		当暂停播放时触发 pause 事件
bindended	EventHandle		当播放到末尾时触发 ended 事件
bindtimeupdate	EventHandle		播放进度变化时触发，event.detail = {currentTime: '当前播放时间'} 。触发频率应该在 250ms 一次
bindfullscreenchange	EventHandle		当视频进入和退出全屏时触发，event.detail = {fullScreen: '当前全屏状态'}
objectFit	String	contain	当视频大小与 video 容器大小不一致时，视频的表现形式，有 contain：包含；fill：填充；cover：覆盖
poster	String		默认控件上的音频封面的图片资源地址，如果 controls 属性值为 false 则设置 poster 无效

src：视频播放的 url 地址。

controls：控制显示播放控件，默认为 true。如果设置为 false，播放器没有操作控件，这时可以自定义播放、暂停、进度调节等按钮。

danmu-btn：设置是否显示弹幕按钮，只在初始化时有效，不能动态变更。开始之后，弹幕信息来自 danmu-list 数组。

autoplay：src 加载视频之后，视频是否自动播放。

objectFit：同图片类似，展示视频的样式，有 contain：包含；fill：填充；cover：覆盖。

bindplay、bindpause、bindended、bindtimeupdate：分别绑定播放事件、暂停事件、播放结束事件和播放进度发生改变事件。

我们可以录制视频，或从相册获取视频来播放，具体使用代码如下所示。开发工具显示效果如图 3-55 所示。手机预览效果如图 3-56 所示。

```
<!--pages/pageView3/pageView3.wxml-->
<!--从相册获取视频-->
<view class="section">
  <video class="videoC"  src="{{src}}"   controls ></video>
    <button bindtap="bindButtonTap">选取视频</button>
</view>

// pages/pageView3/pageView3.js
```

```
    //从相册获取视频
      bindButtonTap: function() {

          var that = this
        wx.chooseVideo({
          sourceType: ['album', 'camera'],
          maxDuration: 60,
          camera: ['front','back'],
          success: function(res) {
            that.setData({
              src: res.tempFilePath
            })
          }
        })
      },
```

图 3-55 开发者工具选择视频

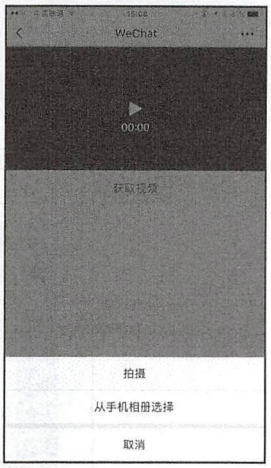


图 3-56 手机预览选择视频

视频绑定所有事件，监控视频的状态，代码如下所示：

```
    <!--pages/pageView3/pageView3.wxml-->
    <!--绑定事件-->
    <!--bindplay、bindpause、bindended、bindtimeupdate-->
    <view class="section">绑定事件</view>
    <video  class="videoC"  src="http://wx.leadingdo.com/videos/test.mp4"
    bindplay="bindplayFun" bindpause="bindpauseFun" bindended="bindendedFun"
bindtimeupdate="bindtimeupdateFun" binderror="videoErrorCallback"></video>

    // pages/pageView3/pageView3.js
    // 视频播放
      bindplayFun: function () {
        console.log("bindplayFun")
      },
      // 视频暂停
      bindpauseFun: function () {
        console.log("bindpauseFun")
      },
      // 视频结束
```

```
    bindendedFun: function () {
      console.log("bindendedFun")
    },
    // 视频进度改变
    bindtimeupdateFun: function () {
      console.log("bindtimeupdateFun")
    },
    // 视频播放异常
    videoErrorCallback: function () {
      console.log("videoErrorCallback")
    },
```

视频播放器的弹幕功能，微信已经封装得非常好，这里我只用了很简单的几个属性，danmu-btn 显示弹幕按钮，danmu-list 存放弹幕的列表数组。

弹幕列表的元素：

```
{
text: '这个视频不错哦',//弹幕文本
color: '#ff0000',//弹幕颜色
time: 1//弹幕出现的时间
}
```

具体使用代码如下所示，显示效果如图 3-57 所示。

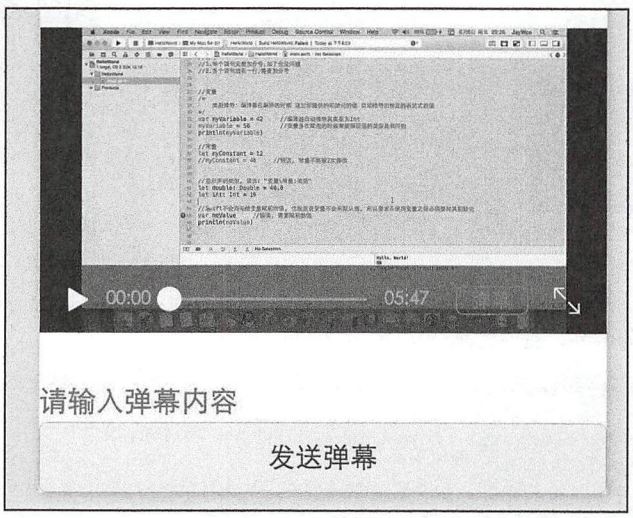


图 3-57　弹幕效果

```
<!--pages/pageView3/pageView3.wxml-->
<!--弹幕效果，发送弹幕-->
<view class="section">
  <video id="myVideo" class="videoC"  src="http://wx.leadingdo.com/videos/test.mp4" danmu-list="{{danmuList}}" enable-danmu danmu-btn controls></video>
  <view class="section">
    <input bindblur="bindInputBlur" placeholder="请输入弹幕内容"/>
    <button bindtap="bindSendDanmu">发送弹幕</button>
  </view>
</view>
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
// pages/pageView3/pageView3.js

// 生成随机颜色
function getRandomColor() {
  let rgb = []
  for (let i = 0; i < 3; ++i) {
    let color = Math.floor(Math.random() * 256).toString(16)
    color = color.length == 1 ? '0' + color : color
    rgb.push(color)
  }
  return '#' + rgb.join('')
}
Page({
  onReady: function (res) {
    this.videoContext = wx.createVideoContext('myVideo')
  },
  inputValue: '',
  data: {
    src: '',
    danmuList: [
      {
        text: '第 1 个弹幕 1s 的时候出现',
        color: '#ff0000',
        time: 1
      },
      {
        text: '第 2 个弹幕 3s 的时候出现',
        color: '#ff00ff',
        time: 3
      }]
  },
  bindInputBlur: function (e) {
    this.inputValue = e.detail.value
  },

  //从相册获取视频
  bindButtonTap: function () {

    var that = this
    wx.chooseVideo({
      sourceType: ['album', 'camera'],
      maxDuration: 60,
      camera: ['front', 'back'],
      success: function (res) {
        that.setData({
          src: res.tempFilePath
        })
      }
    })
  },
  // 视频播放
  bindplayFun: function () {
    console.log("bindplayFun")
  },
  // 视频暂停
  bindpauseFun: function () {
    console.log("bindpauseFun")
  },
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    // 视频结束
    bindendedFun: function () {
      console.log("bindendedFun")
    },
    // 视频进度改变
    bindtimeupdateFun: function () {
      console.log("bindtimeupdateFun")
    },
    // 视频播放异常
    videoErrorCallback: function () {
      console.log("videoErrorCallback")
    },
    bindSendDanmu: function () {
      this.videoContext.sendDanmu({
        text: this.inputValue,
        color: getRandomColor()
      })
    }
  })
```

3.7.4 camera

系统相机，需要用户授权 scope.camera。camera 组件是由客户端创建的原生组件，它的层级是最高的，不能通过 z-index 控制层级，可使用 cover-view、cover-image 覆盖在上面。同一页面只能插入一个 camera 组件。不能在 scroll-view、swiper、picker-view、movable-view 中使用 camera 组件。

使用代码如下所示：

```
<!--pages/pageView4/pageView4.wxml-->
<camera device-position="back" flash="off" binderror="error" style="width: 100%; height: 300px;"></camera>
<button type="primary" bindtap="takePhoto">拍照</button>
<view>预览</view>
<image mode="widthFix" src="{{src}}"></image>
```

camera 组件相关属性如表 3-46 所示。

表 3-46　　camera 组件相关属性

属性	类型	默认值	说明
device-position	String	back	前置或后置，值为 front、back
flash	String	auto	闪光灯，值为 auto、on、off
bindstop	EventHandle		摄像头在非正常终止时触发，如退出后台等情况
binderror	EventHandle		用户不允许使用摄像头时触发

使用相机 API:wx.createCameraContext 进行拍照。使用代码如下所示：

```
// pages/pageView4/pageView4.js
Page({

//拍照
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    takePhoto() {
      const ctx = wx.createCameraContext()
      ctx.takePhoto({
        quality: 'high',
        success: (res) => {
          this.setData({
            src: res.tempImagePath
          })
        }
      })
    },
    error(e) {
      console.log(e.detail)
    }
})
```

3.8 地图（map）

微信小程序地图操作比较简单，API 也很少，使用 map 组件来展示。说到地图，那就先来看基础定位。

定位要用到 wx.getLocation()函数，代码如下所示，详细请参考 API 章的介绍。

```
//index.js
Page({
  data: {
  },
  onLoad: function () {
    wx.getLocation({
      type: 'wgs84', // 默认为 wgs84 返回 gps 坐标，gcj02 返回可用于 wx.openLocation 的坐标
      success: function (res) {
        // success
        var latitude = res.latitude
        var longitude = res.longitude
        var speed = res.speed
        var accuracy = res.accuracy


        console.log("latitude: " + latitude);
        console.log("longitude" + longitude);
        console.log("speed" + speed);
        console.log("accuracy" + accuracy);
      },
      fail: function () {
        // fail
      },
      complete: function () {
        // complete
      }
    })
  }
})
```

地图组件，主要用于在页面上显示地图，进行 LBS 服务或位置指引导航。地图组件的经

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

纬度必填，如果不填经纬度则默认值是北京的经纬度。map 组件的属性，如表 3-47 所示。

表 3-47　　map 属性

属性	类型	默认值	说明
longitude	Number		中心经度
latitude	Number		中心纬度
scale	Number	16	缩放级别，取值范围为 5~18
markers	Array		标记点
covers	Array		即将移除，请使用 markers
polyline	Array		路线
circles	Array		圆
controls	Array		控件
include-points	Array		缩放视野以包含所有给定的坐标点
show-location	Boolean		显示带有方向的当前定位点
bindmarkertap	EventHandle		单击标记点时触发
bindcallouttap	EventHandle		单击标记点对应的气泡时触发
bindcontroltap	EventHandle		单击控件时触发
bindregionchange	EventHandle		视野发生变化时触发
bindtap	EventHandle		单击地图时触发

地图组件使用代码如后几页所示，显示效果如图 3-58 所示。

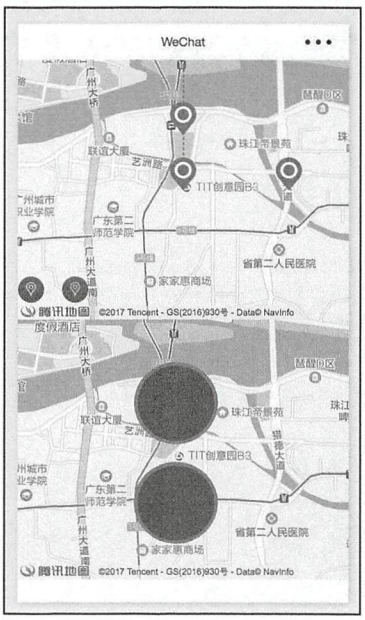


图 3-58　map 组件

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    <!--index.wxml-->
    <map id="map" longitude="113.324520" latitude="23.099994" scale="14" controls=
"{{controls}}" bindcontroltap="controltap" markers="{{markers}}" bindmarkertap=
"markertap" polyline="{{polyline}}" bindregionchange="regionchange"
bindtap="bindtapFun" show-location style="width: 100%; height: 300px;"></map>


    <map id="map" longitude="113.324520" latitude="23.099994" scale="14"
circles="{{circles}}" show-location style="width: 100%; height: 300px;"></map>

    //index.js
    Page({
      data: {
        // markers 标记点数组
        markers: [{
          iconPath: "../resources/location.png",
          id: 0,
          latitude: 23.099994,
          longitude: 113.324520,
          width: 30,
          height: 38
        }, {
          iconPath: "../resources/location.png",
          id: 1,
          latitude: 23.10509,
          longitude: 113.324520,
          width: 30,
          height: 38
        }, {
          iconPath: "../resources/location.png",
          id: 2,
          latitude: 23.099994,
          longitude: 113.334820,
          width: 30,
          height: 38
        }],
        // polyline 路线
        polyline: [{
          points: [{
            longitude: 113.3245211,
            latitude: 23.10229
          }, {
            longitude: 113.324520,
            latitude: 23.21229
          },],
          color: "#FF00DD",
          width: 5,
          dottedLine: true
        }],
        // circles 圆
        circles: [
          {
          latitude: 23.095994,
          longitude: 113.324520,
          color:"#ff5400",
          fillColor:"#5400ff",
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        radius:400,
        strokeWidth:2
        }, {
        latitude: 23.10509,
        longitude: 113.324520,
        color:"#ff5400",
        fillColor:"#5400ff",
        radius:400,
        strokeWidth:2
        }
      ],
      // controls 在地图上显示控件，控件不随着地图移动
      controls: [{
        id: 1,
        iconPath: '../resources/others.png',
        position: {
          left: 0,
          top: 300 - 50,
          width: 30,
          height: 30
        },
        clickable: true
      }, {
        id: 2,
        iconPath: '../resources/others.png',
        position: {
          left: 50,
          top: 300 - 50,
          width: 30,
          height: 30
        },
        clickable: true
      }]
    },
    // 单击标记点时触发
    markertap(e) {
      console.log(e.markerId)
    },
    // 单击控件时触发
    controltap: function (e) {
      console.log(e.controlId)
    },
    // 视野发生变化时触发
    regionchange(e) {
      console.log(e.type)
    },
    // 单击地图时触发
    bindtapFun: function (e) {
      console.log(e)
    },
    //获取当前定位信息
    onLoad: function () {
      wx.getLocation({
        type: 'wgs84', // 默认为 wgs84 返回 gps 坐标，gcj02 返回可用于 wx.openLocation 的坐标
        success: function (res) {
          // success
          var latitude = res.latitude
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        var longitude = res.longitude
        var speed = res.speed
        var accuracy = res.accuracy


        console.log("latitude: " + latitude);
        console.log("longitude" + longitude);
        console.log("speed" + speed);
        console.log("accuracy" + accuracy);
      },
      fail: function () {
        // fail
      },
      complete: function () {
        // complete
      }
    })
  },
})
```

covers 属性即将移除，请使用 markers 替代。markers 标记点用于在地图上显示标记的位置。

markers 数组，每个对象包含以下属性，用来自定义图标和样式，如表 3-48 所示。

表 3-48　　markers 属性

属性	说明	类型	必填	备注
id	标记点 id	Number	否	markers 单击事件回调会返回此 id
latitude	纬度	Number	是	浮点数，范围-90 ~ 90
longitude	经度	Number	是	浮点数，范围-180 ~ 180
title	标注点名	String	否	
iconPath	显示的图标	String	是	项目目录下的图片路径，支持相对路径写法，以'/'开头则表示相对小程序根目录
rotate	旋转角度	Number	否	顺时针旋转的角度，范围 0 ~ 360，默认为 0
alpha	标注的透明度	Number	否	默认 1，不透明
width	标注图标宽度	Number	否	默认为图片实际宽度
height	标注图标高度	Number	否	默认为图片实际高度
callout	自定义标记点上方的气泡窗口	Object	否	{content, color, fontSize, borderRadius, bgColor, padding, boxShadow, display}
label	为标记点旁边增加标签	Object	否	{color, fontSize, content, x, y}，可识别换行符，*x*、*y* 原点是 markers 对应的经纬度
anchor	经纬度在标注图标的锚点，默认底边中点	Object	否	{x, y}，*x* 表示横向(0~1)，*y* 表示竖向(0~1)。{x:0.5, y: 1} 表示底边中点

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

markers 上的气泡 callout，属性如表 3-49 所示。

表 3-49 callout 属性

属性	说明	类型
content	文本	String
color	文本颜色	String
fontSize	文字大小	Number
borderRadius	callout 边框圆角	Number
bgColor	背景色	String
padding	文本边缘留白	Number
display	'BYCLICK':单击显示; 'ALWAYS':常显	String

polyline 数组，指定一系列坐标点，从数组第一项连线至最后一项，数组的每一项可以设置属性，如表 3-50 所示。

表 3-50 polyline 属性

属性	说明	类型	必填	备注
points	经纬度数组	Array	是	[{latitude: 0, longitude: 0}]
color	线的颜色	String	否	8 位十六进制表示，后两位表示 alpha 值，如：#000000AA
width	线的宽度	Number	否	
dottedLine	是否虚线	Boolean	否	默认 false
arrowLine	带箭头的线	Boolean	否	默认 false，开发者工具暂不支持该属性
borderColor	线的边框颜色	String	否	
borderWidth	线的厚度	Number	否	

controls 数组在地图上显示控件，控件不随着地图移动，数组中每一项的属性如表 3-51 所示。

表 3-51 controls 属性

属性	说明	类型	必填	备注
id	控件 id	Number	否	在控件单击事件回调会返回此 id
position	控件在地图的位置	Object	是	控件相对地图位置
iconPath	显示的图标	String	是	项目目录下的图片路径，支持相对路径写法，以'/'开头则表示相对小程序根目录
clickable	是否可单击	Boolean	否	默认不可单击

controls 数组中，position 对象的属性如表 3-52 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 3-52 position 属性

属性	说明	类型
left	距离地图的左边界多远	Number
top	距离地图的上边界多远	Number
width	控件宽度	Number
height	控件高度	Number

circles 数组用于在地图上显示圆，数组中每一项可以设置的属性如表 3-53 所示。

表 3-53 circles 属性

属性	说明	类型	必填	备注
latitude	纬度	Number	是	浮点数，范围-90～90
longitude	经度	Number	是	浮点数，范围-180～180
color	描边的颜色	String	否	8 位十六进制表示，后两位表示 alpha 值，如：#000000AA
fillColor	填充颜色	String	否	8 位十六进制表示，后两位表示 alpha 值，如：#000000AA
radius	半径	Number	是	
strokeWidth	描边的宽度	Number	否	

3.9 画布（canvas）

canvas 定义一块画布，在画布内，使用相关 API：wx.createContext、wx.drawCanvas 进行绘画。详细使用方法可以参考 API 章的介绍。

canvas-id 是画布的唯一标识符，canvas 通过标识符不同来定义多个画布，如果重复定义 canvas-id，页面只显示第一个画布，重复的画布将隐藏不再显示。

canvas 属性，如表 3-54 所示。

表 3-54 canvas 属性

属性	类型	默认值	说明
canvas-id	String		canvas 组件的唯一标识符
disable-scroll	Boolean	false	当在 canvas 中移动时，禁止屏幕滚动以及下拉刷新
bindtouchstart	EventHandle		手指触摸动作开始
bindtouchmove	EventHandle		手指触摸后移动
bindtouchend	EventHandle		手指触摸动作结束
bindtouchcancel	EventHandle		手指触摸动作被打断，如来电提醒、弹窗

非卖品，仅供非商业用途或交流学习使用

续表

属性	类型	默认值	说明
bindlongtap	EventHandle		手指长按 500ms 之后触发，触发了长按事件后进行移动不会触发屏幕的滚动
binderror	EventHandle		当发生错误时触发 error 事件，detail = {errMsg: 'something wrong'}

canvas 标签默认宽度为 300px、高度为 225px。开发者可以通过 style="width: 300px; height: 200px; background-color :#eeeeee" 或者 class = "section"来定义画布的样式。

使用代码如下所示：

```
    <!--index.wxml-->

    <!--通过 style 来设置画布样式-->
    <canvas style="width: 300px; height: 200px; background-color :#eeeeee"
canvas-id="firstCanvas"></canvas>

    <!--通过 class 来设置画布样式-->
    <canvas class = "section" canvas-id="secondCanvas"></canvas>

    <!--因为 canvas-id 与前一个 canvas 重复，该 canvas 不会显示，并调用 canvasIdError-
Callback 事件返回错误-->
    <canvas style="width: 100%; height: 200px; background-color :#eeffee"
canvas-id="secondCanvas" binderror="canvasIdErrorCallback"></canvas>

    /**index.wxss**/

    .section {
      margin-top: 20px;
      width: 100%;
      height: 200px;
      background-color: #eef;
    }

    //index.js
    Page({

      // 画布返回错误
      canvasIdErrorCallback: function (e) {
        console.error(e.detail)
      },

      onReady: function (e) {

        // 使用 wx.createContext 获取绘图上下文 context
        var context = wx.createContext()

        context.setStrokeStyle("#00ff00")
        context.setLineWidth(5)
        context.rect(0, 0, 200, 200)
        context.stroke()
        context.setStrokeStyle("#ff0000")
        context.setLineWidth(2)
```

非卖品，仅供非商业用途或交流学习使用

```
      context.moveTo(160, 100)
      context.arc(100, 100, 60, 0, 2 * Math.PI, true)
      context.moveTo(140, 100)
      context.arc(100, 100, 40, 0, Math.PI, false)
      context.moveTo(85, 80)
      context.arc(80, 80, 5, 0, 2 * Math.PI, true)
      context.moveTo(125, 80)
      context.arc(120, 80, 5, 0, 2 * Math.PI, true)
      context.stroke()

      // 调用 wx.drawCanvas，通过 canvasId 指定在哪张画布上绘制，通过 actions 指定
      //绘制行为
      wx.drawCanvas({
        canvasId: 'firstCanvas',
        actions: context.getActions() // 获取绘图动作数组
      })
    }
  })
```

3.10 客服会话按钮（contact-button）

contact-button 用于在页面上显示一个客服会话按钮，用户单击该按钮后会进入客服会话。

contact-button 属性，如表 3-55 所示。

表 3-55 contact-button 属性

属性	类型	默认值	说明
size	Number	18	会话按钮大小，有效值 18~27，单位为 px
type	String	default-dark	会话按钮的样式类型，有效值 default-dark、default-light
session-from	String		用户从该按钮进入会话时，开发者将收到带上本参数的事件推送。本参数可用于区分用户进入客服会话的来源

使用代码如下所示：

```
<contact-button
  type="default-light"
  size="20"
  session-from="weapp"
>
</contact-button>
```

客服会话具体使用方法，将在 5.7 节客服消息中详细介绍。

3.11 开放数据（open-data）

从基础库 1.4.0 开始支持开放数据（open-data），open-data 用于展示微信开放的数

非卖品，仅供非商业用途或交流学习使用

据。只有当前用户在此群内才能拉取到群名称。open-data 的属性如表 3-56 所示。

表 3-56　　open-data 属性

属性	类型	默认值	说明
type	String		开放数据类型
open-gid	String		当 type="groupName" 时生效，群 id

type 的有效值，如表 3-57 所示。

表 3-57　　type 有效值

值	说明	最低版本
groupName	拉取群名称	1.4.0

使用代码如下所示：

```
<open-data type="groupName" open-gid="xxxxxx"></open-data>
```

非卖品，仅供非商业用途或交流学习使用

第四章

API

微信小程序 MINA 框架，提供了丰富的原生 API，可以方便地调用微信的各种功能，如获取用户信息、本地存储、地理位置、支付功能等。本章主要讲网络 API、媒体 API、文件 API、数据缓存 API、位置 API、设备 API、界面 API 的相关使用。

API 以 wx.开头，如未特殊约定，一般都接收一个 OBJECT 作为参数。其中 wx.on 开头的 API 是监听某个事件发生的 API 接口，接收一个 CALLBACK 函数。当该事件触发时，会调用 CALLBACK 函数。

OBJECT 中可以指定 success、fail、complete 来接收接口调用结果，如表 4-1 所示。

表 4-1　　OBJECT 中指定回调

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

API 主要用于逻辑层的开发，实现原生 App 具有的一些功能，例如利用网络 API 获取丰富的内容、通过媒体 API 实现多样化信息交流，等等。下面将一一介绍各 API 的具体使用。

4.1　网络

使用网络 API 时，需要先登录微信公众平台，在小程序“设置-开发设置-服务器域名”中设置好一个通用域名，如图 4-1 所示。

小程序可以与指定的域名进行网络通信，包括 HTTPS 请求（wx.request）、WebSocket 通信（wx.connectSocket）、上传文件（wx.uploadFile）和下载文件（wx.downloadFile）。目前与网络有关的 API 有下面 10 个接口：

- wx.request（OBJECT）：发起网络请求，可以获取到 json 内容，用于小程序显示。
- wx.uploadFile（OBJECT）：上传文件，可以将小程序本地资源（图片、视频、文档等）上传到服务器。

- wx.downloadFile（OBJECT）：下载文件，可以将服务器上的资源，下载到本地使用。
- wx.connectSocket（OBJECT）：创建 WebSocket 连接，WebSocket 协议是基于 TCP 的一种新的协议，可以实现小程序与服务器全双工（full-duplex）通信，实现即时通信。
- wx.onSocketOpen（CALLBACK）：用于监听 WebSocket 打开。
- wx.onSocketError（CALLBACK）：用于监听 WebSocket 错误。
- wx.sendSocketMessage（OBJECT）：用于发送 WebSocket 消息。
- wx.onSocketMessage（CALLBACK）：用于接收 WebSocket 消息。
- wx.closeSocket()：关闭 WebSocket 连接。
- wx.onSocketClose（CALLBACK）：用于监听 WebSocket 关闭。

图 4-1　配置服务器域名

网络请求的 referer 是不可以设置的，格式固定为：

https://servicewechat.com/{appid}/{version}/page-frame.html，其中{appid}为小程序的 appid，{version}为小程序的版本号，版本号为 0 表示开发版。

4.1.1　wx.request（OBJECT）发起请求

wx.request 用来发起一个 HTTPS 请求。一个微信小程序只能同时发起 5 个网络请求。request 的默认超时时间和最大超时时间都是 60s。该方法的 OBJECT 参数可以理解为一个字典，里面包含很多“key-value”值。request 的 OBJECT 参数说明，如表 4-2 所示。

表 4-2　　request 的 OBJECT 参数

参数	类型	必填	说明
url	String	是	开发者服务器接口地址
data	Object/String	否	请求的参数

续表

参数	类型	必填	说明
header	Object	否	设置请求的 header，header 中不能设置 referer
method	String	否	默认为 GET，有效值：OPTIONS、GET、HEAD、POST、PUT、DELETE、TRACE、CONNECT
dataType	String	否	默认为 json。如果设置了 dataType 为 json，则会尝试对响应的数据做一次 JSON.parse
success	Function	否	收到开发者服务成功返回的回调函数，res = {data: '开发者服务器返回的内容'}
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

- url 是请求的服务器 https 地址，不能有端口。如果 url 后缀不是具体的文件，必须以“/”结束。例如：https://wx.leadingdo.com/serverTime 是错误的，https://wx.leadingdo.com/serverTime/才能正确使用。
- data：数据说明，最终发送给服务器的数据是 String 类型，如果传入的 data 不是 String 类型，会被转换成 String。转换根据 header 字段设置的“content-type”类型进行。如果'content-type': 'application/json'，会对数据进行 JSON 序列化。如果'content-type': ' application/x-www-form-urlencoded'，会将数据进行 urlencoded 编码。
- method：请求类型，value 值必须为大写（例如：GET、POST）。

wx.request 使用代码，如下所示：

```
// 4.1.1 发起请求
  testButtonClick1: function () {

    //request 请求
    wx.request({

      //请求地址
      url: 'https://wx.leadingdo.com/serverTime/Default.aspx',

      //请求参数
      data: {
        uName: '刘明洋',
        uid: '001'
      },

      //设置 header 信息
      header: {
        'content-type': 'application/x-www-form-urlencoded'
      },

      //请求方式
      method: 'POST',

      //收到数据 json 处理
```

```
        dataType: 'json',

        //成功之后回调
        success: function (res) {
          console.log("request success:" + res.data['errorCode'])
          console.log("request success:" + res.data['msg'])
          console.log("request success:" + res.data['serverTime'])


        },

        //失败回调
        fail: function (err) {
          console.log("request fail:" + err)
        },

        //结束回调
        complete: function (err) {
          console.log("request complete:" + err)
        }
      })
    },
```

如果使用 method: 'POST'，需要设置 header 信息为'content-type':'application/x-www-form-urlencoded'。如果使用 GET 方式的话，header 设置为'content-type': 'application/json'，否则服务器可能接收不到 data 参数。

从基础库 1.4.0 开始，wx.request 接口具有返回值 requestTask 对象，通过 requestTask 的 abort 方法，可以中断任务请求。

代码如下所示：

```
const requestTask = wx.request({
  url: 'test.php', //仅为示例，并非真实的接口地址
  data: {
  x: '' ,
  y: ''
  },
  header: {
      'content-type': 'application/json'
  },
  success: function(res) {
    console.log(res.data)
  }
})

requestTask.abort() // 取消请求任务
```

关于 wx.request 几点注意事项如下：

1. content-type 默认为 'application/json'；
2. 开发者工具 0.10.102800 版本，header 的 content-type 设置异常；
3. 客户端的 HTTPS TLS 版本为 1.2，但 Android 的部分机型还未支持 TLS 1.2，所以请确保 HTTPS 服务器的 TLS 版本支持 1.2 及以下版本；

4. 要注意 method 的 value 必须为大写（例如：GET）；

5. url 中不能有端口；

6. request 的默认超时时间和最大超时时间都是 60s；

7. request 的最大并发数是 10 个；

8. 网络请求的 referer 是不可以设置的，格式固定为 https://servicewechat.com/{appid}/{version}/page-frame.html，其中 {appid} 为小程序的 appid，{version} 为小程序的版本号，版本号为 0 表示为开发版。

客户端的 HTTPS TLS 版本为 1.2，但 Android 的部分机型还不支持 TLS 1.2，所以请确保 HTTPS 服务器的 TLS 版本支持 1.2 及以下版本，如图 4-2 所示。新版开发者工具增加了 https 检查功能，开发者可使用此功能直接本地避开 ssl 协议版本检查，但是此功能对真机无效。

服务器请选用认可的证书，开发者可以通过腾讯云官网提供的检测工具（进入腾讯云官网，选择产品栏中的 SSL 证书，再选择苹果 ATS 检测），来检测证书的有效性。如图 4-3 所示。

图 4-2 开发环境不校验请求域名以及 TLS 版本

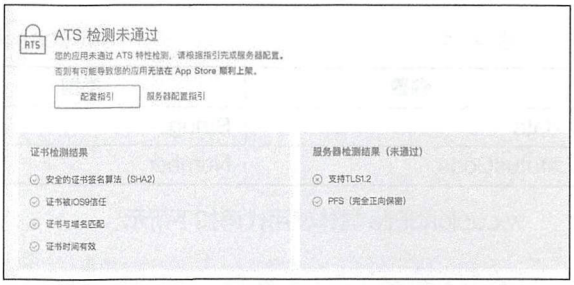


图 4-3 证书检测

如果证书未通过检测，可以单击“配置指引”，来对服务器证书进行相关配置。这里可以选择不同的 Web 服务器，有以下 4 种。

1. Nginx 证书配置；

2. Apache 证书配置；

3. Tomcat 证书配置；

4. IIS 证书配置。

4.1.2 上传、下载

wx.uploadFile（OBJECT）将本地资源上传到开发者服务器，最大并发限制是 10 个，默认超时时间和最大超时时间都是 60s。wx.uploadFile（OBJECT）可以通过 wx.chooseImage 等接口获取到一个本地资源的临时文件路径，并通过此接口将本地资源上传到指定服务器。客户

端发起一个 HTTPS POST 请求，其中 content-type 为 multipart/form-data。uploadFile 的 OBJECT 参数说明，如表 4-3 所示。

表 4-3　　uploadFile 的 OBJECT 参数

参数	类型	必填	说明
url	String	是	开发者服务器 url
filePath	String	是	要上传文件资源的路径
name	String	是	文件对应的 key，开发者在服务器端通过这个 key 可以获取到文件二进制内容
header	Object	否	HTTP 请求 header，header 中不能设置 referer
formdata	Object	否	HTTP 请求中其他额外的 formdata
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

wx.uploadFile 也是通过 success 回调，获取返回参数，如表 4-4 所示。

表 4-4　　success 返回参数

参数	类型	说明
data	String	开发者服务器返回的数据
statusCode	Number	HTTP 状态码

wx.uploadFile 具体使用代码如下所示：

```
// 4.1.2 上传
  testButtonClick2: function () {

    //选择照片
    wx.chooseImage({
      success: function (res) {

        //获取临时路径
        var tempFilePaths = res.tempFilePaths

        //上传
        wx.uploadFile({

          //上传接口
          url: 'https://wx.leadingdo.com/uploadFile/Default.aspx',

          //上传文件路径
          filePath: tempFilePaths[0],

          //文件对应的 key，方便服务器获取
          name: 'file',

          //设置 header 信息
          header: {
```

```
            'content-type':'multipart/form-data'
          },

          //额外上传一些信息
          formData: {
            'user': '刘明洋'
          },

          //成功之后回调
          success: function (res) {
            console.log("request success:" + res.data)

          },

          //失败回调
          fail: function (err) {
            console.log("request fail:" + err)
          },

          //结束回调
          complete: function (err) {
            console.log("request complete:" + err)
          }
        })
      }
    })
  },
```

从基础库版本 1.4.0 开始，wx.uploadFile 方法返回一个 uploadTask 对象，通过 uploadTask 对象的 onProgressUpdate 方法可以监听到上传进度变化。abort 方法可以取消上传任务，使用代码如下所示：

```
const uploadTask = wx.uploadFile({
  url: 'http://example.weixin.qq.com/upload', //仅为示例，非真实的接口地址
  filePath: tempFilePaths[0],
  name: 'file',
  formData:{        'user': 'test'
  },
  success: function(res){
    var data = res.data        //do something
  }
})

 uploadTask.onProgressUpdate((res) => {
  console.log('上传进度', res.progress)
  console.log('已经上传的数据长度', res.totalBytesSent)
  console.log('预期需要上传的数据总长度', res.totalBytesExpectedToSend)
})

uploadTask.abort() // 取消上传任务
```

wx.downloadFile（OBJECT）API 可以下载文件资源到本地，最大并发限制是 10 个，默认超时时间和最大超时时间都是 60s。

客户端直接发起一个 HTTP GET 请求，下载成功之后，会返回文件的临时存储路径。该临时路径，在小程序本次启动期间可以正常使用，如果下次启动小程序想继续使用该文件，需

要主动调用 wx.saveFile API 进行持久化保存，这样在小程序下次启动时才能访问得到。downloadFile 的 OBJECT 参数，如表 4-5 所示。

表 4-5　　downloadFile 的 OBJECT 参数

参数	类型	必填	说明
url	String	是	下载资源的 url
header	Object	否	HTTP 请求 header
success	Function	否	下载成功后以 tempFilePath 的形式传给页面，res = {tempFilePath: '文件的临时路径'}
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

wx.downloadFile 具体使用代码如下所示：

```
// 4.1.2 下载
  testButtonClick3: function () {

    wx.downloadFile({

      //请求地址
      url: 'https://wx.leadingdo.com/images/book1.png',

      //设置 header 信息
      header: {

      },

      //成功之后回调
      success: function (res) {
        console.log("downloadFile success:" + res.tempFilePath)

      },

      //失败回调
      fail: function (err) {
        console.log("downloadFile fail:" + err)
      },

      //结束回调
      complete: function (err) {
        console.log("downloadFile complete:" + err)
      }
    })
  },
```

从基础库 1.4.0 开始，wx.downloadFile 方法返回一个 downloadTask 对象，通过 downloadTask 对象的 onProgressUpdate 方法可以监听到下载进度变化。abort 方法可以取消下载任务，使用代码如下所示：

```
const downloadTask = wx.downloadFile({
```

```
  url: 'http://example.com/audio/123', //仅为示例，并非真实的资源
  success: function(res) {
wx.playVoice({
      filePath: res.tempFilePath
 })
  }
})

downloadTask.onProgressUpdate((res) => {
  console.log('下载进度', res.progress)
  console.log('已经下载的数据长度', res.totalBytesWritten)
  console.log('预期需要下载的数据总长度', res.totalBytesExpectedToWrite)
})

downloadTask.abort() // 取消下载任务
```

4.1.3 Websocket

1）wx.connectSocket（OBJECT）创建一个 WebSocket 连接，一个微信小程序只能同时有一个 WebSocket 连接。如果当前已存在一个 WebSocket 连接，会自动关闭该连接，并重新创建一个 WebSocket 连接。连接默认超时时间和最大超时时间都是 60s。connectSocket 的 OBJECT 参数说明如表 4-6 所示。

表 4-6　　connectSocket 的 OBJECT 参数

参数	类型	必填	说明
url	String	是	开发者服务器接口地址，必须是 wss 协议，且域名必须是后台配置的合法域名
data	Object	否	请求的数据
header	Object	否	HTTP header，header 中不能设置 referer
method	String	否	默认是 GET，有效值为：OPTIONS、GET、HEAD、POST、PUT、DELETE、TRACE、CONNECT
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

2）wx.onSocketOpen（CALLBACK）：监听 WebSocket 连接打开事件。connectSocket 创建 WebSocket 成功之后，Socket 通道打开，会调用 onSocketOpen 方法。

3）wx.onSocketError（CALLBACK）：监听 WebSocket 错误。connectSocket 创建失败时，调用该方法。

4）wx.sendSocketMessage（OBJECT）：通过 WebSocket 连接发送数据，需要先创建 wx.connectSocket，并在 wx.onSocketOpen 回调之后才能发送。sendSocketMessage 的 OBJECT 参数说明如表 4-7 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 4-7 sendSocketMessage 的 OBJECT 参数

参数	类型	必填	说明
data	String/ArrayBuffer	是	需要发送的内容
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

5）wx.onSocketMessage（CALLBACK）：接收服务器消息事件。onSocketMessage 的 CALLBACK 返回参数如表 4-8 所示。

表 4-8 onSocketMessage 的 CALLBACK 参数

参数	类型	说明
data	String/ArrayBuffer	服务器返回的消息

6）wx.closeSocket()：关闭 WebSocket 连接。必须在调用了 wx.onSocketOpen，确定打开了 WebSocket 连接后，才可以使用 wx.closeSocket()关闭连接。

7）wx.onSocketClose（CALLBACK）：监听 WebSocket 关闭。

关于 WebSocket 的 7 个 API，使用代码如下所示，具体可以参考代码中的备注信息。

```
//index.js

var socketOpen = false
var socketMsgQueue = []

Page({
  data: {
  },
  onLoad: function () {
    console.log('onLoad')
  },

  // 4.1.3 wx.connectSocket
  socketButtonClick: function () {

    //创建 WebSocket 连接
      console.log('创建 WebSocket 连接！')
    wx.connectSocket({

      //socket 合法域名
      url: 'wss://wx.leadingdo.com',

      //请求的数据
      data: {
        x: '',
        y: ''
      },

      //HTTP header , header 中不能设置 referer
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    header: {
      'content-type': 'application/json'
    },

    //请求方式
    method: "GET",

    //成功之后回调
    success: function (res) {
      console.log("connectSocket success")

    },

    //失败回调
    fail: function (err) {
      console.log("connectSocket fail:")
    },

    //结束回调
    complete: function (err) {
      console.log("connectSocket complete:")
    }
  })


  //监听打开事件
  wx.onSocketOpen(function (res) {
    console.log('WebSocket连接已打开！')

    //修改socketOpen标记
    socketOpen = true

    //判断数组中是否存在内容，如果存在内容，发送到服务器
    for (var i = 0; i < socketMsgQueue.length; i++) {
      //发送消息
      sendSocketMessage(socketMsgQueue[i])
    }
    socketMsgQueue = []
  })

  //监听WebSocket错误
  wx.onSocketError(function (res) {
    console.log('WebSocket连接打开失败！')
  })

  //接收消息事件
  wx.onSocketMessage(function (res) {
    console.log('收到服务器内容：' + res.data)
  })

  //监听WebSocket关闭
  wx.onSocketClose(function (res) {
    console.log('WebSocket 已关闭！')
  })
},

//发送消息
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
  socketButtonClick1: function () {

    //如果 socket 是开启状态，执行发现，否则先显示存放到数组中
    if (socketOpen) {

      //发送消息
      wx.sendSocketMessage({
        data: msg,

        //成功之后回调
        success: function (res) {
          console.log("sendSocketMessage success")

        },

        //失败回调
        fail: function (err) {
          console.log("sendSocketMessage fail:")
        },

        //结束回调
        complete: function (err) {
          console.log("sendSocketMessage complete:")
        }
      })
    } else {
      socketMsgQueue.push(msg)
    }
  },

  //关闭 WebSocket
  socketButtonClick2: function () {

    //如果 socket 开启，执行关闭操作
    if (socketOpen) {
      wx.closeSocket()
    }
    socketOpen = false
    console.log('WebSocket 关闭！')
  }
})

<!--index.wxml-->
<view class="section2">4.1.3 WebSocket </view>
<button  bindtap="socketButtonClick" class="section">创建并打开 WebSocket
</button>
<button  bindtap="socketButtonClick1" class="section">发送数据
</button>
<button  bindtap="socketButtonClick2" class="section">关闭 WebSocket
</button>

/**index.wxss**/
.section {
  margin-top: 10px;
}
.section2 {
  margin-top: 50px;
}
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

8）SocketTask

WebSocket 任务，可通过 wx.connectSocket() 接口创建返回。从基础库 1.7.0 开始支持，低版本需做兼容处理。

- SocketTask.send(OBJECT)，通过 WebSocket 连接发送数据。

SocketTask.send 方法的 OBJECT 参数说明如表 4-9 所示。

表 4-9　　SocketTask.send 方法的 OBJECT 参数

参数	类型	必填	说明
data	String/ArrayBuffer	是	需要发送的内容
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

- SocketTask.close(OBJECT)，关闭 WebSocket 连接。

SocketTask.close 方法的 OBJECT 参数说明如表 4-10 所示。

表 4-10　　SocketTask.close 方法的 OBJECT 参数

参数	类型	必填	说明
code	Number	否	一个数字值，表示关闭连接的状态号。如果这个参数没有被指定，默认的取值是 1000（表示正常连接关闭）
reason	String	否	一个可读的字符串，表示连接被关闭的原因。这个字符串必须是不长于 123 字节的 UTF-8 文本（不是字符）
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

- SocketTask.onOpen(CALLBACK)监听 WebSocket 连接打开事件。
- SocketTask.onClose(CALLBACK)监听 WebSocket 连接关闭事件。
- SocketTask.onError(CALLBACK)监听 WebSocket 错误。

这 3 个方法的 CALLBACK 返回参数如表 4-11 所示。

表 4-11　　CALLBACK 返回参数

参数	类型	说明
errMsg	String	错误信息

- SocketTask.onMessage(CALLBACK)监听 WebSocket 接收到的服务器消息事件。

SocketTask.onMessage 方法的 CALLBACK 返回参数如表 4-12 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 4-12 SocketTask.onMessage 方法的 CALLBACK 返回参数

参数	类型	说明
data	String/ArrayBuffer	服务器返回的消息

4.2 媒体

媒体 API 可以方便地实现选取照片、预览照片、音频的录制和播放、音乐的播放、视频的播放。利用这些 API 可以开发一些视频播放器、音频播放器。

4.2.1 图片

与图片相关的 API 有 4 个，可以实现选择相册中的图片或使用拍照功能，获取图片的信息，以及预览图片的功能等。

1）wx.chooseImage（OBJECT），可以实现从相册中选择图片，或者使用拍照功能。chooseImage 的 OBJECT 参数说明如表 4-13 所示。

表 4-13 chooseImage 的 OBJECT 参数

参数	类型	必填	说明
count	Number	否	最多可以选择的图片张数，默认 9
sizeType	StringArray	否	original 原图，compressed 压缩图，默认二者都有
sourceType	StringArray	否	album 从相册选图，camera 使用相机，默认二者都有
success	Function	是	成功则返回图片的本地文件路径列表 tempFilePaths
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

成功之后，返回 tempFilePaths 文件的临时路径，在小程序本次启动期间可以正常使用，如需保存，需要主动调用 wx.saveFile，在小程序下次启动时才能访问。

wx.chooseImage 具体代码如下所示：

```
// 选择
  testButtonClick11: function () {

    wx.chooseImage({
      count: 6, // 默认 9
      sizeType: ['original', 'compressed'], // 可以指定是原图还是压缩图，默认二者都有
      sourceType: ['album', 'camera'], // 可以指定来源是相册还是相机，默认二者都有
      success: function (res) {
        console.log("chooseImage success:" + res.tempFilePaths)
        // 返回选定照片的本地文件路径列表，tempFilePath 可以作为 img 标签的 src 属性显示图片
        var tempFilePaths = res.tempFilePaths
      },
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        //失败回调
        fail: function (err) {
          console.log("chooseImage fail:" + err)
        },

        //结束回调
        complete: function (err) {
          console.log("chooseImage complete:" + err)
        }
      })
    },
```

2）wx.previewImage（OBJECT）：预览网络图片。其OBJECT参数说明如表4-14所示。

表4-14　　previewImage的OBJECT参数

参数	类型	必填	说明
current	String	否	当前显示图片的链接，不填则默认为urls的第一张
urls	StringArray	是	需要预览的图片链接列表
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

wx.previewImage具体使用代码如下所示：

```
    //  预览
      testButtonClick12: function () {

        wx.previewImage({

          // 当前显示图片的https链接
          current: 'https://wx.leadingdo.com/images/book3.png',

          // 需要预览的图片https链接列表
          urls: ["https://wx.leadingdo.com/images/book1.png", "https://wx.leadingdo.
com/images/book2.png", "https://wx.leadingdo.com/images/book3.png"],

          success: function (res) {
            console.log("previewImage success:")
          },
          //失败回调
          fail: function (err) {
            console.log("previewImage fail:" + err)
          },

          //结束回调
          complete: function (err) {
            console.log("previewImage complete:" + err)
          }
        })
      },
```

3）wx.getImageInfo（OBJECT）：用来获取图片信息，getImageInfo的OBJECT参

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

数说明如表 4-15 所示。

表 4-15 getImageInfo 的 OBJECT 参数

参数	类型	必填	说明
src	String	是	图片的路径，可以是相对路径、临时文件路径、存储文件路径、网络图片路径
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

获取成功之后，回调 success，success 可以通过返回参数获取图片的宽度、高度和具体路径。success 返回参数说明如表 4-16 所示。

表 4-16 获取图片成功 success 的返回参数

参数	类型	说明
width	Number	图片宽度，单位为 px
height	Number	图片高度，单位为 px
path	String	返回图片的本地路径

读取相对路径图片信息、读取相册图片信息，具体代码如下所示：

```
//  获取图片信息
  testButtonClick13: function () {

    //读取相对路径图片
    wx.getImageInfo({
      src: 'images/book.png',
      success: function (res) {
        console.log(res.width)
        console.log(res.height)
        console.log(res.path)
      },
      //失败回调
      fail: function (err) {
        console.log("getImageInfo fail:" + err)
      },

      //结束回调
      complete: function (err) {
        console.log("getImageInfo complete:" + err)
      }
    })

    //选择相册图片，获取信息
    wx.chooseImage({
      success: function (res) {
        wx.getImageInfo({
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
            //获取选择的照片数组，取第 1 张
            src: res.tempFilePaths[0],
            success: function (res) {
              console.log(res.width)
              console.log(res.height)
              console.log(res.path)
            },

            //失败回调
            fail: function (err) {
              console.log("getImageInfo fail:" + err)
            },

            //结束回调
            complete: function (err) {
              console.log("getImageInfo complete:" + err)
            }
          })
        },

        //失败回调
        fail: function (err) {
          console.log("chooseImage fail:" + err)
        },

        //结束回调
        complete: function (err) {
          console.log("chooseImage complete:" + err)
        }
      })
    },
```

4）wx.saveImageToPhotosAlbum(OBJECT)，保存图片到系统相册。

从基础库 1.2.0 开始支持，保存图片到系统相册，该功能需要用户授权接口 wx.authorize(OBJECT)，授权 scope.writePhotosAlbum 权限，关于授权可以参考 5.3 节内容。

saveImageToPhotosAlbum 的 OBJECT 参数说明如表 4-17 所示。

表 4-17　　saveImageToPhotosAlbum 的 OBJECT 参数说明

参数	类型	必填	说明
filePath	String	是	图片文件路径，可以是临时文件路径也可以是永久文件路径，不支持网络图片路径
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

从 success 返回的参数 errMsg，可以看到调用的结果。wx.saveImageToPhotosAlbum 的使用代码如下所示：

```
<!--index.wxml-->
 <button bindtap="testButtonClick14" class="section"> 保存图片到系统相册 </button>
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
//index.js
//下载的图片位置
var imagePath = ''

//page 下面实现按钮事件，保存图片到系统相册
function saveImage() {

  wx.saveImageToPhotosAlbum({
    filePath: imagePath,

    //成功之后回调
    success: function (res) {
      console.log("saveImageToPhotosAlbum success:" + res.tempFilePath)

    },

    //失败回调
    fail: function (err) {
      console.log("saveImageToPhotosAlbum fail:" + err)
    },

    //结束回调
    complete: function (err) {
      console.log("saveImageToPhotosAlbum complete:" + err)
    }
  })
}

  // 保存图片到系统相册
  testButtonClick14: function () {
    wx.downloadFile({
      //请求地址
      url: 'https://wx.leadingdo.com/images/book1.png',
      //设置 header 信息
      header: {
      },
      //成功之后回调
      success: function (res) {

        imagePath = res.tempFilePath

        console.log("Path:" + imagePath)

        // 可以通过 wx.getSetting 先查询一下用户是否授权了 "scope.writePhotosAlbum"
        // 这个 scope
        wx.getSetting({
          success(res) {

            //没有授权的，需要先授权
            if (!res.authSetting['scope.writePhotosAlbum']) {
              wx.authorize({
                scope: 'scope.writePhotosAlbum',
                success() {

                  // 授权成功之后，保存图片
                  saveImage()
                }
```

非卖品，仅供非商业用途或交流学习使用

```
            })
          } else {
            //已授权过的，直接保存图片
            saveImage()
          }
        }
      })
    }
  })
},
```

4.2.2 录音

录音 API 实现了微信录音功能，主要由开启录音“wx.startRecord（OBJECT）”和结束录音“wx.stopRecord()”组成。

1）wx.startRecord（OBJECT）：开始录音。可以调用 wx.stopRecord，或者录音超过 1 分钟时自动结束录音，返回录音文件的临时文件路径。临时文件路径在下次打开小程序时无效，如果需要保存，可以调用 wx.saveFile。startRecord 的 OBJECT 参数说明如表 4-18 所示。

表 4-18　　startRecord 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	录音成功后调用，返回录音文件的临时文件路径，res = {tempFilePath: '录音文件的临时路径'}
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

2）wx.stopRecord()：主动调用停止录音。

录音使用代码如下所示：

```
//  开始录音
  testButtonClick21: function () {

    //开始录音
    wx.startRecord({
      success: function (res) {
        console.log("startRecord success"+res)
        var tempFilePath = res.tempFilePath
      },
      fail: function (res) {
        //录音失败
        console.log("startRecord fail"+res)
      },
      complete: function (res) {
        //录音完成
        console.log("startRecord complete"+res)
      },
```

```
    })

    //设置延时 20s 后调用 wx.stopRecord()关闭录音
    setTimeout(function () {
      //结束录音
      wx.stopRecord()
    }, 20000)
  },

  //  结束录音
  testButtonClick22: function () {
    //主动结束录音
    wx.stopRecord()
  },
```

开启录音功能之后，导航栏会显示“录音中”，这里可以根据前面章节介绍的组件使用、数据绑定来自定义录制界面的显示效果。自己可以去尝试一下。

4.2.3 语音播放控制

语音播放控制 API 主要实现本地语音文件的播放、暂停和停止 3 个 API。

1）wx.playVoice（OBJECT）：开始播放语音，同时只允许一个语音文件正在播放，如果前一个语音文件还没播放完，将中断前一个语音播放。playVoice 的 OBJECT 参数说明如表 4-19 所示。

表 4-19　　playVoice 的 OBJECT 参数说明

参数	类型	必填	说明
filePath	String	是	需要播放的语音文件的文件路径
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

2）wx.pauseVoice()：暂停正在播放的语音。再次调用 wx.playVoice 播放同一个文件时，会从暂停处开始播放。如果想从头开始播放，需要先调用 wx.stopVoice 结束播放。

3）wx.stopVoice()：结束播放语音。

语音播放控制的使用，代码如下所示：

```
  //  开始播放
  testButtonClick31: function () {
    wx.playVoice({

      //语音文件路径
      filePath: voicePath,

      //成功回调
      success: function () {
        console.log("playVoice success")
```

```
    },

    //失败回调
    fail: function () {
      console.log("playVoice fail")
    },

    //完成
    complete: function () {
      console.log("playVoice complete")
    }
  })
},

//  暂停播放
testButtonClick32: function () {
  wx.pauseVoice()
},

//  继续播放
testButtonClick33: function () {
  this.testButtonClick31();//调用播放事件
},

//  结束播放
testButtonClick34: function () {
  wx.stopVoice()
},
```

设置的语音文件路径 filePath:voicePath，voicePath 是录音结束后的临时文件目录，也可以使用已经保存了的音频文件路径。

4.2.4 音乐播放控制

音乐播放控制，可以播放一个 url 链接的音乐文件，使用后台播放器播放音乐，对于微信客户端来说，只能同时有一个后台音乐正在播放。当用户离开小程序后，音乐将暂停播放；当用户单击“显示在聊天顶部”时，音乐不会暂停播放；当用户使用其他小程序占用音乐播放器时，原有小程序内的音乐将停止播放。

1）wx.getBackgroundAudioPlayerState（OBJECT）：获取后台音乐播放状态。getBackgroundAudioPlayerState 的 OBJECT 参数说明如表 4-20 所示。

表 4-20　　getBackgroundAudioPlayerState 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数，可以获取到后台是否有音乐正在播放，如果有音乐正在播放，可

以获取播放音乐的时长、播放位置、下载进度等信息。success 返回参数说明如表 4-21 所示。

表 4-21　　success 返回参数说明

参数	说明
duration	选定音频的时长（单位为 s），只有在当前有音乐播放时返回
currentPosition	选定音频的播放位置（单位为 s），只有在当前有音乐播放时返回
status	播放状态（2：没有音乐在播放；1：播放中；0：暂停中）
downloadPercent	音频的下载进度（整数，80 代表 80%），只有在当前有音乐播放时返回
dataUrl	歌曲数据链接，只有在当前有音乐播放时返回

获取后台音乐播放状态，具体使用代码如下所示：

```
// 获取后台音乐播放状态
  testButtonClick41: function () {

    wx.getBackgroundAudioPlayerState({
      success: function (res) {

        //读取 success 返回参数
        var status = res.status
        var dataUrl = res.dataUrl
        var currentPosition = res.currentPosition
        var duration = res.duration
        var downloadPercent = res.downloadPercent
      }
    })
  },
```

2）wx.playBackgroundAudio（OBJECT）：使用后台播放器播放音乐，当用户使用其他小程序占用音乐播放器时，原有小程序内的音乐将停止播放。playBackgroundAudio 的 OBJECT 参数说明，如表 4-22 所示。

表 4-22　　playBackgroundAudio 的 OBJECT 参数说明

参数	类型	必填	说明
dataUrl	String	是	音乐链接
title	String	否	音乐标题
coverImgUrl	String	否	音乐封面链接
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

3）wx.pauseBackgroundAudio()：暂停播放音乐。

4）wx.seekBackgroundAudio（OBJECT）：控制音乐播放进度，OBJECT 参数说明如表 4-23 所示。

表 4-23　　seekBackgroundAudio 的 OBJECT 参数说明

参数	类型	必填	说明
position	Number	是	音乐播放位置（单位为 s）
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

5）wx.stopBackgroundAudio()：停止播放音乐。

6）wx.onBackgroundAudioPlay（CALLBACK）：监听音乐播放。

7）wx.onBackgroundAudioPause（CALLBACK）：监听音乐暂停。

8）wx.onBackgroundAudioStop（CALLBACK）：监听音乐停止。

音乐播放控制相关 API 的使用代码如下所示：

```
//  使用后台播放器播放音乐
  testButtonClick42: function () {
    wx.playBackgroundAudio({
      dataUrl: 'https://wx.leadingdo.com/audios/test.mp3', //音乐地址
      title: '测试音乐',//音乐标题
      coverImgUrl: 'https://wx.leadingdo.com/audios/test.jpg',//音乐封面

      //成功回调
      success: function () {
        console.log("playBackgroundAudio success")
      },

      //失败回调
      fail: function () {
        console.log("playBackgroundAudio fail")
      },

      //完成
      complete: function () {
        console.log("playBackgroundAudio complete")
      }
    })

    // 监听音乐播放
    wx.onBackgroundAudioPlay(function () {
      console.log("音乐开始播放！")
    })

    // 监听音乐暂停
    wx.onBackgroundAudioPause(function () {
      console.log("音乐暂停！")
    })

    // 监听音乐停止
    wx.onBackgroundAudioStop(function () {
      console.log("音乐停止！")
    })
  },
```

```
// 暂停播放音乐
testButtonClick43: function () {
  wx.pauseBackgroundAudio()
},

// 控制音乐播放进度
testButtonClick44: function () {
  wx.seekBackgroundAudio({
    position: 30,

    //成功回调
    success: function () {
      console.log("seekBackgroundAudio success")
    },

    //失败回调
    fail: function () {
      console.log("seekBackgroundAudio fail")
    },

    //完成
    complete: function () {
      console.log("seekBackgroundAudio complete")
    }
  })
},

// 停止播放音乐
testButtonClick45: function () {
  wx.stopBackgroundAudio()
},
```

9）音频组件控制

在第三章讲 audio 组件的时候，我们也用到了音频组件控制 API，通过本 API 可以控制 audio 组件。

wx.createAudioContext（audioId）：创建并返回 audio 上下文 audioContext 对象，audioContext 通过 audioId 与一个<audio>组件绑定，操作对应的<audio>组件。createAudioContext 对象的方法如表 4-24 所示。

表 4-24　　createAudioContext 的方法

方法	参数	说明
setSrc	src	音频的地址
play	无	播放
pause	无	暂停
seek	position	跳转到指定位置，单位为 s

音频组件控制，具体使用代码如下所示：

```
<!--index.wxml-->
```

```
    <view class="section">音乐播放组件 </view>
    <audio poster="{{poster}}" name="{{name}}" author="{{author}}" src="{{src}}"
id="myAudio" controls loop></audio>
    <button type="primary" bindtap="audioPlay">播放</button>
    <button type="primary" bindtap="audioPause">暂停</button>
    <button type="primary" bindtap="audio16">设置当前播放时间为 16s</button>
    <button type="primary" bindtap="audioStart">回到开头</button>

    //index.js
    Page({
      data: {

        poster: 'https://wx.leadingdo.com/audios/test.jpg',
        name: '此时此刻',
        author: '许巍',
        src: 'https://wx.leadingdo.com/audios/test.mp3',
      },
    // 音乐播放组件
      onReady: function (e) {
        // 使用 wx.createAudioContext 获取 audio 上下文 context
        this.audioCtx = wx.createAudioContext('myAudio')
      },

      // 开始播放
      audioPlay: function () {
        this.audioCtx.play()
      },

      //暂停
      audioPause: function () {
        this.audioCtx.pause()
      },

      //从 16s 开始播放
      audio16: function () {
        this.audioCtx.seek(16)
      },

      //重新播放
      audioStart: function () {
        this.audioCtx.seek(0)
      }
    })
```

10）背景音频播放管理

从基础库 1.2.0 开始，小程序增加了获取全局唯一的背景音频管理器 backgroundAudioManager，backgroundAudioManager 对象的属性如表 4-25 所示。

表 4-25　　backgroundAudioManager 的属性

属性	类型	必填	说明
duration	Number	是	当前音频的时长（单位为 s），只有在当前有合法的 src 时返回
currentTime	Number	是	当前音频的播放位置（单位为 s），只有在当前有合法的 src 时返回

续表

属性	类型	必填	说明
paused	Boolean	是	当前是否在暂停或停止状态，true 表示暂停或停止，false 表示正在播放
src	String	否	音频的数据源，默认为空字符串，当设置了新的 src 时，会自动开始播放，目前支持的格式有 m4a、aac、mp3、wav
startTime	Number	否	音频开始播放的位置（单位为 s）
buffered	Number	是	音频缓冲的时间点，仅保证当前播放时间点到此时间点内容已缓冲
title	String	否	音频标题，用作原生音频播放器音频标题。原生音频播放器中的分享功能分享出去的卡片标题也将使用该值
epname	String	否	专辑名，原生音频播放器中的分享功能分享出去的卡片简介也将使用该值
singer	String	否	歌手名，原生音频播放器中的分享功能分享出去的卡片简介也将使用该值
coverImgUrl	String	否	封面图链接，用作原生音频播放器背景图。原生音频播放器中的分享功能，分享出去的卡片配图及背景也将使用该图
webUrl	String	否	页面链接，原生音频播放器中的分享功能分享出去的卡片简介也将使用该值

backgroundAudioManager 对象的方法如表 4-26 所示。

表 4-26　　backgroundAudioManager 的方法

方法	参数	说明
play		播放
pause		暂停
stop		停止
seek	position	跳转到指定位置，单位为 s
onCanplay	callback	背景音频进入可以播放状态，但不保证后面可以流畅播放
onPlay	callback	背景音频播放事件
onPause	callback	背景音频暂停事件
onStop	callback	背景音频停止事件
onEnded	callback	背景音频自然播放结束事件
onTimeUpdate	callback	背景音频播放进度更新事件
onPrev	callback	用户在系统音乐播放面板单击上一曲事件（iOS only）
onNext	callback	用户在系统音乐播放面板单击下一曲事件（iOS only）
onError	callback	背景音频播放错误事件
onWaiting	callback	音频加载中事件，当音频因为数据不足，需要停下来加载时会触发

errCode 说明，如表 4-27 所示。

表 4-27 errCode 说明

errCode	说明
10001	系统错误
10002	网络错误
10003	文件错误
10004	格式错误
-1	未知错误

wx.getBackgroundAudioManager 使用代码如下所示：

```
<!--index.wxml-->
<view class="section">背景音频管理器 </view>
<button type="primary" bindtap="backAudioPlay">播放</button>
<button type="primary" bindtap="backAudioPause">暂停</button>
<button type="primary" bindtap="backAudio16">设置当前播放时间为 16s</button>
<button type="primary" bindtap="backAudioStart">停止</button>

//index.js
//////// //////// //////// ////////背景音频管理器
  //开始播放
  backAudioPlay: function () {
    const backgroundAudioManager = wx.getBackgroundAudioManager()

    backgroundAudioManager.title = '此时此刻'
    backgroundAudioManager.epname = '此时此刻'
    backgroundAudioManager.singer = '汪峰'
    backgroundAudioManager.coverImgUrl = 'http://y.gtimg.cn/music/photo_new/
T002R300x300M000003rsKF44GyaSk.jpg?max_age=2592000'
    backgroundAudioManager.src = 'http://ws.stream.qqmusic.qq.com/
M500001VfvsJ21xFqb.mp3?guid=ffffffff82def4af4b12b3cd9337d5e7&uin=346897220&vkey=
6292F51E1E384E061FF02C31F716658E5C81F5594D561F2E88B854E81CAAB7806D5E4F103E55D
33C16F3FAC506D1AB172DE8600B37E43FAD&fromtag=46' // 设置了 src 之后会自动播放

    backgroundAudioManager.play()

    //监听回调事件

    //背景音频进入可以播放状态，但不保证后面可以流畅播放
    backgroundAudioManager.onCanplay = (
      console.log("backgroundAudioManager ==== onCanplay")
    )

    //背景音频播放事件
    backgroundAudioManager.onPlay = (
      console.log("backgroundAudioManager ==== onPlay")
    )

    //背景音频暂停事件
    backgroundAudioManager.onPause = (
      console.log("backgroundAudioManager ==== onPause")
    )
```

```
    //背景音频停止事件
    backgroundAudioManager.onStop = (
      console.log("backgroundAudioManager ==== onStop")
    )

    //背景音频自然播放结束事件
    backgroundAudioManager.onEnded = (
      console.log("backgroundAudioManager ==== onEnded")
    )

    //背景音频播放进度更新事件
    backgroundAudioManager.onTimeUpdate = (
      console.log("backgroundAudioManager ==== onTimeUpdate")
    )

    //用户在系统音乐播放面板单击上一曲事件（iOS only）
    backgroundAudioManager.onPrev = (
      console.log("backgroundAudioManager ==== onPrev")
    )

    //用户在系统音乐播放面板单击下一曲事件（iOS only）
    backgroundAudioManager.onNext = (
      console.log("backgroundAudioManager ==== onNext")
    )

    //背景音频播放错误事件
    backgroundAudioManager.onError = (
      console.log("backgroundAudioManager ==== onError")
    )

    //音频加载中事件，当音频因为数据不足，需要停下来加载时会触发
    backgroundAudioManager.onWaiting = (
      console.log("backgroundAudioManager ==== onWaiting")
    )
  },

  //暂停
  backAudioPause: function () {

    const backgroundAudioManager = wx.getBackgroundAudioManager()
    backgroundAudioManager.pause()
  },

  //从 16s 开始播放
  backAudio16: function () {

    const backgroundAudioManager = wx.getBackgroundAudioManager()
    backgroundAudioManager.seek(16)
  },

  //停止播放
  backAudioStart: function () {
    const backgroundAudioManager = wx.getBackgroundAudioManager()
    backgroundAudioManager.stop()
  },
```

4.2.5 视频和视频组件控制

视频 API 主要播放本地视频或拍摄录制的视频，通过 wx.chooseVideo 来选取视频文件。视频组件控制和音频组件控制类似，可以创建一个 video 上下文 videoContext 对象，videoContext 通过 videoId 与一个<video>组件绑定，操作对应的<video>组件。

1）wx.chooseVideo（OBJECT）：选取本地视频或拍摄视频，成功之后 success 返回视频的临时文件路径。chooseVideo 的 OBJECT 参数说明如表 4-28 所示。

表 4-28 chooseVideo 的 OBJECT 参数说明

参数	类型	必填	说明
sourceType	StringArray	否	album 从相册选视频，camera 使用相机拍摄，默认为：['album', 'camera']
maxDuration	Number	否	拍摄视频的最长时间，单位为 s。最长支持 60s
camera	String	否	使用前置摄像头还是后置摄像头。front: 前置；back: 后置，默认 back
success	Function	否	接口调用成功，返回视频文件的临时文件路径，详见返回参数说明
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

成功之后，success 返回参数说明如表 4-29 所示。

表 4-29 success 返回参数

参数	说明
tempFilePath	选定视频的临时文件路径
duration	选定视频的时间长度
size	选定视频的数据量大小
height	返回选定视频的长
width	返回选定视频的宽

tempFilePath 是文件的临时路径，在小程序本次启动期间可以正常使用，如需保存，需要主动调用 wx.saveFile，在小程序下次启动时才能访问。

视频 API 的使用代码如下所示：

```
    <!--index.wxml-->
    <view class="section2">4.2.5 视频与视频组件 </view>
    <video src="{{videoParh}}"></video>
    <button bindtap="testButtonClick51" class="section">拍摄视频或从手机相册中选取视频</button>

    //index.js
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
Page({
  data: {
    videoParh:''
  },
//  拍摄视频或从手机相册中选取视频
  testButtonClick51: function () {
    var that = this
    wx.chooseVideo({
      sourceType: ['album', 'camera'],
      maxDuration: 60,
      camera: 'back',
      success: function (res) {
        that.setData({
          videoParh: res.tempFilePath
        })
        console.log(res.duration)
        console.log(res.size)
        console.log(res.height)
        console.log(res.width)
      }
    })
  },
})
```

2）wx.saveVideoToPhotosAlbum(OBJECT)：保存视频到相册。和 wx.saveImageToPhotosAlbum 使用类似，需要用户授权（scope.writePhotosAlbum）。

saveVideoToPhotosAlbum 的 OBJECT 参数如表 4-30 所示。

表 4-30　　saveVideoToPhotosAlbum 的 OBJECT 参数

参数	类型	必填	说明
filePath	String	是	视频文件路径，可以是临时文件路径也可以是永久文件路径
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

从 success 返回参数 errMsg，可以看到调用的结果。wx.saveVideoToPhotosAlbum 的使用代码如下所示：

```
wx.saveVideoToPhotosAlbum({
    filePath:''
success(res) {
 }
})
```

3）wx.createVideoContext（videoId）：创建并返回 video 上下文 videoContext 对象，videoContext 通过 videoId 与一个 <video> 组件绑定，操作对应的 <video> 组件。创建 videoContext 对象的方法如表 4-31 所示。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 4-31　　createVideoContext 方法列表

方法	参数	说明
play	无	播放
pause	无	暂停
seek	position	跳转到指定位置，单位为 s
sendDanmu	danmu	发送弹幕，danmu 包含两个属性 text、color
playbackRate	rate	设置倍速播放，支持的倍率有 0.5/0.8/1.0/1.25/1.5
requestFullScreen	无	进入全屏
exitFullScreen	无	退出全屏

视频组件具体使用代码如下所示：

```
<!--index.wxml-->

<view class="section2">视频组件</view>
  <video id="myVideo" src="https://wx.leadingdo.com/audios/ccNews.mp4"
enable-danmu danmu-btn controls></video>
    <input bindblur="bindInputBlur"/>
    <button bindtap="bindSendDanmu">发送弹幕</button>
<button type="primary" bindtap="videoPlay">播放</button>
<button type="primary" bindtap="videoPause">暂停</button>
<button type="primary" bindtap="video16">设置当前播放时间为 16s</button>

//index.js
//  拍摄视频或从手机相册中选取视频
  testButtonClick51: function () {
    var that = this
    wx.chooseVideo({
      sourceType: ['album', 'camera'],
      maxDuration: 60,
      camera: 'back',
      success: function (res) {
        that.setData({
          videoParh: res.tempFilePath
        })
      }
    })
  },

  //  视频组件
  inputValue: '',//弹幕内容

  //在 input 文本框中输入内容事件
  bindInputBlur: function(e) {
    this.inputValue = e.detail.value
  },

  //发送弹幕
  bindSendDanmu: function () {
    this.videoContext.sendDanmu({
      text: this.inputValue,
      color: getRandomColor()
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    })
  },

  // 开始播放
  videoPlay: function () {
    this.videoContext.play()
  },

  //暂停
  videoPause: function () {
    this.videoContext.pause()
  },

  //从16s开始播放
  video16: function () {
    this.videoContext.seek(16)
  }
```

4）wx.createCameraContext(this)

相机组件控制，从基础库 1.6.0 开始支持，低版本需做兼容处理。wx.createCameraContext 创建并返回 camera 上下文 cameraContext 对象，cameraContext 与页面的 <camera> 组件绑定（一个页面只能有一个 camera），操作对应的 <camera/> 组件。在自定义组件下，第一个参数传入组件实例 this，以操作组件内 <camera/> 组件。

- cameraContext 对象的方法列表，如表 4-32 所示。

表 4-32　　cameraContext 对象的方法列表

方法	参数	说明
takePhoto	OBJECT	拍照，可指定质量，成功则返回图片
startRecord	OBJECT	开始录像
stopRecord	OBJECT	结束录像，成功则返回封面与视频

- takePhoto 的 OBJECT 参数说明，如表 4-33 所示。

表 4-33　　takePhoto 的 OBJECT 参数说明

参数	类型	必填	说明
quality	String	否	成像质量，可选值为 high、normal、low，默认值为 normal
success	Function	否	接口调用成功的回调函数，res = { tempImagePath }
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

- startRecord 的 OBJECT 参数说明，如表 4-34 所示。

表 4-34　　startRecord 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

续表

参数	类型	必填	说明
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）
timeoutCallback	Function	否	超过 30s 或页面 onHide 时会结束录像，res = { tempThumbPath, tempVideoPath }

- stopRecord 的 OBJECT 参数说明，如表 4-35 所示。

表 4-35 stopRecord 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数，res = { tempThumbPath, tempVideoPath }
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

4.3 文件

文件 API 主要实现了对文件的保存、获取已保存文件列表、获取单个文件的信息、删除文件、打开文件。下面将一一介绍各 API。

1）wx.saveFile（OBJECT）：保存文件到本地。前面章节讲的获取图片和视频，返回的都是临时路径，可以通过此 API 进行保存并做永久化处理。本地文件存储的大小限制为 10MB。saveFile 的 OBJECT 参数说明如表 4-36 所示。

表 4-36 saveFile 的 OBJECT 参数说明

参数	类型	必填	说明
tempFilePath	String	是	需要保存的文件的临时路径
success	Function	否	返回文件的保存路径，res = {savedFilePath: '文件的保存路径'}
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

保存文件到本地，具体代码如下所示：

```
    <!--index.wxml-->
      <button  bindtap="testButtonClick1" class="section">获取照片，保存到本地
</button>

    //index.js
    Page({
      data: {
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    savedFilePath: ''
  },
  onLoad: function () {
    console.log('onLoad')
  },

  //选取照片，保存到本地
  testButtonClick1: function () {

    //选取照片
    wx.chooseImage({
      success: function (res) {

        //获得临时路径
        var tempFilePaths = res.tempFilePaths

        //保存到本地
        wx.saveFile({
          tempFilePath: tempFilePaths[0],
          success: function (res) {

            //记录保存后的路径
            this.setData({
              savedFilePath: res.tempFilePath
            })
          }
        })
      }
    })
  }
})
```

2）wx.getFileInfo(OBJECT)：获取文件信息，从基础库 1.4.0 开始支持。getFileInfo 的 OBJECT 参数说明如表 4-37 所示。

表 4-37　　getFileInfo 的 OBJECT 参数说明

参数	类型	必填	说明
filePath	String	是	本地文件路径
digestAlgorithm	String	否	计算文件摘要的算法，默认值 md5，有效值：md5、sha1
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-38 所示。

表 4-38　　success 返回参数说明

参数	类型	说明
size	Number	文件大小，单位为 KB
digest	String	按照传入的 digestAlgorithm 计算得出的文件摘要
errMsg	String	调用结果

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

getFileInfo 使用代码如下所示：

```
wx.getFileInfo({
  filePath:'',
  success(res) {
    console.log(res.size)
    console.log(res.digest)
  }
})
```

3）wx.getSavedFileList（OBJECT）：获取本地已保存的文件列表。getSavedFileList 的 OBJECT 参数说明如表 4-39 所示。

表 4-39　　getSavedFileList 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数，返回结果见 success 返回参数说明
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-40 所示。

表 4-40　　success 返回参数说明

参数	类型	说明
errMsg	String	接口调用结果
fileList	Object Array	文件列表

fileList 是一个数组，数组中每个文件包含的具体属性，如表 4-41 所示。

表 4-41　　fileList 数组对象属性

参数	类型	说明
filePath	String	文件的本地路径
createTime	Number	文件保存时的时间戳，从 1970/01/01 08:00:00 到当前时间的秒数
size	Number	文件大小，单位为 KB

具体使用代码如下所示：

```
//获取本地已保存的文件列表
  testButtonClick2: function () {
    wx.getSavedFileList({
      success: function (res) {
        console.log(res.fileList)
        console.log(res.errMsg)

        for (var obj in res.fileList) {
          console.log(res.fileList[obj].createTime)
          console.log(res.fileList[obj].filePath)
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
          console.log(res.fileList[obj].size)

          //保存一个文件路径，获取信息使用
          oneFilePath = res.fileList[obj].filePath
        }
      },
      fail: function (res) {
        console.log('获取文件列表失败')
      },
      complete: function (res) {
        console.log('获取文件列表完成')
      }
    })
  },
```

4）wx.getSavedFileInfo（OBJECT）：获取本地文件的文件信息。getSavedFileInfo的OBJECT参数说明如表4-42所示。

表4-42 getSavedFileInfo的OBJECT参数说明

参数	类型	必填	说明
filePath	String	是	文件路径
success	Function	否	接口调用成功的回调函数，返回结果见success返回参数说明
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success返回参数说明如表4-43所示。

表4-43 success返回参数说明

参数	类型	说明
errMsg	String	接口调用结果
size	Number	文件大小，单位为KB
createTime	Number	文件保存时的时间戳，从1970/01/01 08:00:00到当前时间的秒数

获取文件信息，具体代码如下所示：

```
//获取文件信息
  testButtonClick3: function () {

    wx.getSavedFileInfo({
      filePath: oneFilePath, //文件路径
      success: function (res) {
        console.log(res.errMsg)
        console.log(res.size)
        console.log(res.createTime)
      },
      fail: function (res) {
        console.log('获取文件信息失败')
      },
      complete: function (res) {
        console.log('获取文件信息完成')
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        }
      })
    },
```

5）wx.removeSavedFile（OBJECT）：删除本地存储的文件。removeSavedFile 的 OBJECT 参数说明如表 4-44 所示。

表 4-44　　removeSavedFile 的 OBJECT 参数说明

参数	类型	必填	说明
filePath	String	是	需要删除的文件路径
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

删除文件使用代码如下所示：

```
//删除文件
  testButtonClick4: function () {

    wx.removeSavedFile({
      filePath: oneFilePath,
      success: function (res) {
        console.log('删除文件成功')
      },
      fail: function (res) {
        console.log('删除文件失败')
      },
      complete: function (res) {
        console.log('删除文件完成'+res)
      }
    })
  }
```

6）wx.openDocument（OBJECT）：使用新页面打开文件，支持格式：doc、xls、ppt、pdf、docx、xlsx、pptx。openDocument 的 OBJECT 参数说明如表 4-45 所示。

表 4-45　　openDocument 的 OBJECT 参数说明

参数	说明	必填	说明
filePath	String	是	文件路径，可通过 downFile 获得
fileType	String	否	文件类型，指定文件类型打开文件，有效值 doc、xls、ppt、pdf、docx、xlsx、pptx
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

打开文件，具体代码如下所示：

```
//打开文件
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
testButtonClick5: function () {

  //显示加载
  wx.showNavigationBarLoading()
  //下载文件
  wx.downloadFile({
    url: '文件路径名称',

    //下载成功
    success: function (res) {

      // 获取临时路径
      var filePath = res.tempFilePath

      //打开文件
      wx.openDocument({
        filePath: filePath,
        success: function (res) {
          console.log('打开文件成功')
        },
        fail: function (res) {
          console.log('打开文件失败')
        },
        complete: function (res) {
          console.log('打开文件完成' + res)
        }
      })
    },
    fail: function (res) {
      console.log('下载失败')
    },
    complete: function (res) {
      console.log('下载完成' + res)

      //取消加载状态
      wx.hideNavigationBarLoading()
    }
  })
}
```

4.4 数据缓存

每个小程序都有自己的缓存空间，本地缓存空间最大为 10MB。我们可以将一些信息缓存到本地，方便以后使用。

本节主要介绍数据的存储、读取、删除以及获取缓存空间信息、清空缓存空间。每一个动作都对应着同步方法和异步方法。

1）wx.setStorage（OBJECT）：异步存储数据，指定 key 和 data。如果存储空间已经存在 key 对应的内容，再次存储则覆盖以前的内容。setStorage 的 OBJECT 参数说明如表 4-46 所示。

非卖品，仅供非商业用途或交流学习使用

表 4-46　　setStorage 的 OBJECT 参数说明

参数	类型	必填	说明
key	String	是	本地缓存中指定的 key
data	Object/String	是	需要存储的内容
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
// 异步存储(key-data)
  testButtonClick1: function () {

    wx.setStorage({
      key: "key",
      data: "value",
      //成功之后回调
      success: function (res) {
        console.log("setStorage success:" + res)
      },
      //失败回调
      fail: function (err) {
        console.log("setStorage fail:" + err)
      },
      //结束回调
      complete: function (err) {
        console.log("setStorage complete:" + err)
      }
    })
  },
```

2）wx.setStorageSync（KEY，DATA）：同步将 data 存储在本地缓存中指定的 key 中，会覆盖掉原来该 key 对应的内容。setStorageSync 的参数说明如表 4-47 所示。

表 4-47　　setStorageSync 的参数说明

参数	类型	必填	说明
key	String	是	本地缓存中指定的 key
data	Object/String	是	需要存储的内容

具体代码使用如下所示：

```
// 同步存储(key-data)
  testButtonClick2: function () {
    try {
      wx.setStorageSync('key', 'value')
    } catch (e) {
      console.log("同步存储失败")
    }
  },
```

3）wx.getStorage（OBJECT）：异步从本地缓存中读取指定的 key 对应的内容。getStorage 的 OBJECT 参数说明如表 4-48 所示。

表 4-48　　getStorage 的 OBJECT 参数说明

参数	类型	必填	说明
key	String	是	本地缓存中指定的 key
success	Function	是	接口调用成功的回调函数，res = {data: key 对应的内容}
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

具体使用代码如下所示：

```
// 异步读取(key-data)
  testButtonClick3: function () {
    wx.getStorage({
      key: 'key',
      //成功
      success: function (res) {
        console.log(res.data)
      },
      //失败回调
      fail: function (err) {
        console.log("getStorage fail:" + err)
      },
      //结束回调
      complete: function (err) {
        console.log("getStorage complete:" + err)
      }
    })
  },
```

4）wx.getStorageSync（KEY）：同步从本地缓存中读取指定的 key 对应的内容。使用代码如下所示：

```
// 同步读取(key-data)
  testButtonClick4: function () {
    try {
      var value = wx.getStorageSync('key')
      if (value) {
        console.log("value : " + value)
      }
    } catch (e) {
      console.log("同步读取失败")
    }
  },
```

5）wx.getStorageInfo（OBJECT）：异步获取当前 storage 的相关信息。getStorageInfo 的 OBJECT 参数说明如表 4-49 所示。

表 4-49　　getStorageInfo 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	是	接口调用成功的回调函数，详见返回参数说明
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

通过 success 返回参数，用户可以知道存储空间所有的 key、当前占用空间大小以及空间限制大小，success 返回参数说明如表 4-50 所示。

表 4-50　　success 返回参数说明

参数	类型	说明
keys	String Array	当前 storage 中所有的 key
currentSize	Number	当前占用的空间大小，单位为 KB
limitSize	Number	限制的空间大小，单位为 KB

使用代码如下所示：

```
// 异步获取当前 storage 的相关信息
  testButtonClick5: function () {
    wx.getStorageInfo({
      //成功之后回调
      success: function (res) {
        console.log(res.keys)
        console.log(res.currentSize)
        console.log(res.limitSize)
      },
      //失败回调
      fail: function (err) {
        console.log("getStorageInfo fail:" + err)
      },
      //结束回调
      complete: function (err) {
        console.log("getStorageInfo complete:" + err)
      }
    })
  },
```

6）wx.getStorageInfoSync：同步获取当前 storage 的相关信息。使用代码如下所示：

```
// 同步获取当前 storage 的相关信息
  testButtonClick6: function () {
    try {
      var res = wx.getStorageInfoSync()
      console.log(res.keys)
      console.log(res.currentSize)
      console.log(res.limitSize)
    } catch (e) {
      console.log("同步获取当前 storage 的相关信息失败")
    }
  },
```

7）wx.removeStorage（OBJECT）：异步从本地缓存中删除指定的 key 对应的内容。removeStorage 的 OBJECT 参数说明如表 4-51 所示。

表 4-51　　removeStorage 的 OBJECT 参数说明

参数	类型	必填	说明
key	String	是	本地缓存中指定的 key
success	Function	是	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

具体使用代码如下所示：

```
// 异步删除(key-data)
  testButtonClick7: function () {
    wx.removeStorage({
      key: 'key',
      //成功
      success: function (res) {
        console.log(res.data)
      },
      //失败回调
      fail: function (err) {
        console.log("removeStorage fail:" + err)
      },
      //结束回调
      complete: function (err) {
        console.log("removeStorage complete:" + err)
      }
    })
  },
```

8）wx.removeStorageSync（KEY）：同步从本地缓存中删除指定的 key 对应的内容。使用代码如下所示：

```
// 同步删除(key-data)
  testButtonClick8: function () {
    try {
      wx.removeStorageSync('key')
    } catch (e) {
      console.log("同步删除失败")
    }
  },
```

9）wx.clearStorage()：异步清空本地数据缓存。使用代码如下所示：

```
// 异步清空本地数据缓存
  testButtonClick9: function () {
    wx.clearStorage()
  },
```

10）wx.clearStorageSync()：同步清空本地数据缓存。使用代码如下所示：

```
// 同步清空本地数据缓存
  testButtonClick10: function () {
    try {
      wx.clearStorageSync()
    } catch (e) {
      console.log("同步清空本地数据缓存失败")
    }
  },
```

4.5 位置

位置 API 可以在小程序中实现 LBS 定位相关的功能。

4.5.1 获取位置

1）wx.getLocation（OBJECT）：获取当前的地理位置、速度。当用户离开小程序后，此接口无法调用；当用户单击“显示在聊天顶部”时，此接口可继续调用。getLocation 的 OBJECT 参数说明如表 4-52 所示。

表 4-52 getLocation 的 OBJECT 参数说明

参数	类型	必填	说明
type	String	否	默认为 wgs84 返回 gps 坐标，gcj02 返回可用于 wx.openLocation 的坐标
success	Function	是	接口调用成功的回调函数，返回内容详见返回参数说明
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

用户通过 success 返回参数可以获取到经纬度、速度等参数，具体说明如表 4-53 所示。

表 4-53 success 返回参数说明

参数	说明
latitude	纬度，浮点数，范围为-90～90，负数表示南纬
longitude	经度，浮点数，范围为-180～180，负数表示西经
speed	速度，浮点数，单位为 m/s
accuracy	位置的精确度
altitude	高度，单位为 m
verticalAccuracy	垂直精度，单位为 m（Android 无法获取，返回 0）
horizontalAccuracy	水平精度，单位为 m

具体使用代码如下所示：

```
// 获取位置
  testButtonClick1: function () {
    wx.getLocation({
      type: 'wgs84',

      //成功
      success: function (res) {
        var latitude = res.latitude
        var longitude = res.longitude
        var speed = res.speed
        var accuracy = res.accuracy

        console.log('latitude' + latitude)
        console.log('longitude' + longitude)
        console.log('speed' + speed)
        console.log('accuracy: ' + accuracy)
      },

      //失败回调
      fail: function (err) {
        console.log("getLocation fail:" + err)
      },

      //结束回调
      complete: function (err) {
        console.log("getLocation complete:" + err)
      }
    })
  },
```

2）wx.chooseLocation（OBJECT）：打开地图选择位置。chooseLocation 的 OBJECT 参数说明如表 4-54 所示。

表 4-54　　chooseLocation 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	是	接口调用成功的回调函数，返回内容详见返回参数说明
cancel	Function	否	用户取消时调用
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

用户通过 success 返回参数可以获取到经纬度、速度等参数，具体说明如表 4-55 所示。

表 4-55　　success 返回参数说明

参数	说明
name	位置名称
address	详细地址
latitude	纬度，浮点数，范围为-90～90，负数表示南纬
longitude	经度，浮点数，范围为-180～180，负数表示西经

具体使用代码如下所示：

```
// 打开地图选择位置
  testButtonClick2: function () {

    wx.chooseLocation({
      //成功
      success: function (res) {
        var name = res.name
        var address = res.address
        var latitude = res.latitude
        var longitude = res.longitude

        console.log('name' + name)
        console.log('address' + address)
        console.log('latitude' + latitude)
        console.log('longitude' + longitude)
      },
      //取消
      cancel: function (err) {
        console.log("chooseLocation cancel:" + err)
      },
      //失败回调
      fail: function (err) {
        console.log("chooseLocation fail:" + err)
      },

      //结束回调
      complete: function (err) {
        console.log("chooseLocation complete:" + err)
      }
    })
  },
```

4.5.2 查看位置

wx.openLocation（OBJECT）：使用微信内置地图查看位置。openLocation 的 OBJECT 参数说明如表 4-56 所示。

表 4-56　　openLocation 的 OBJECT 参数说明

参数	类型	必填	说明
latitude	Float	是	纬度，范围为-90～90，负数表示南纬
longitude	Float	是	经度，范围为-180～180，负数表示西经
scale	Int	否	缩放比例，范围 5～18，默认为 18
name	String	否	位置名称
address	String	否	详细地址
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

具体使用代码如下所示：

```
// 使用微信内置地图查看位置
  testButtonClick3: function () {

    //获取当前位置
    wx.getLocation({
      type: 'gcj02', //返回可以用于wx.openLocation的经纬度
      success: function (res) {
        var latitude = res.latitude
        var longitude = res.longitude
        var name = res.name
        var address = res.address

        //使用微信内置地图查看位置
        wx.openLocation({
          latitude: latitude,//纬度
          longitude: longitude,//经度
          scale: 18,//缩放比例
          name: name,//位置名称
          address: address,//详细地址
          //成功
          success: function (res) {
            console.log("openLocation cancel:" + err)
          },
          //失败回调
          fail: function (err) {
            console.log("openLocation fail:" + err)
          },

          //结束回调
          complete: function (err) {
            console.log("openLocation complete:" + err)
          }
        })
      }
    })
  },
```

4.5.3 地图组件控制

wx.createMapContext（mapId）：创建并返回 map 上下文 mapContext 对象，mapContext 通过 mapId 与一个 <map> 组件绑定，操作对应的 <map> 组件。mapContext 对象的方法列表如表 4-57 所示。

表 4-57　　mapContext 对象的方法列表

方法	参数	说明
getCenterLocation	OBJECT	获取当前地图中心的经纬度，返回的是 gcj02 坐标系，可以用于 wx.openLocation
moveToLocation	无	将地图中心移动到当前定位点，需要配合<map>组件的 show-location 使用

续表

方法	参数	说明
translateMarker	OBJECT	平移 marker，带动画
includePoints	OBJECT	缩放视野，展示所有经纬度
getRegion	OBJECT	获取当前地图的视野范围
getScale	OBJECT	获取当前地图的缩放级别

getCenterLocation 的 OBJECT 参数说明如表 4-58 所示。

表 4-58　　getCenterLocation 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数，res = { longitude: "经度", latitude: "纬度"}
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

具体使用代码如下所示：

```
<!--index.wxml-->
<view class="section2">地图组件控制 </view>
<map id="myMap" show-location />
<button type="primary" bindtap="getCenterLocation">获取位置</button>
<button type="primary" bindtap="moveToLocation">移动位置</button>

/**index.wxss**/
.section {
  margin-top: 10px;
}
.section2 {
  margin-top: 50px;
}

//index.js
Page({
  data: {
  },
  onLoad: function () {
    console.log('onLoad')
  },

  // 地图组件控制
  onReady: function (e) {
    // 使用 wx.createMapContext 获取 map 上下文
    this.mapCtx = wx.createMapContext('myMap')
  },

  //获取位置
  getCenterLocation: function () {
    this.mapCtx.getCenterLocation({
      success: function (res) {
        console.log(res.longitude)
```

```
        console.log(res.latitude)
      }
    })
  },

  //移动
  moveToLocation: function () {
    this.mapCtx.moveToLocation()
  }
})
```

4.6 设备

设备 API 主要包括获取系统信息、网络状态、重力加速度、罗盘数据、拨打电话、扫码、剪贴板、蓝牙等相关设备信息的功能。

下面将分别介绍各 API 的功能以及使用方法。

4.6.1 系统信息

1）wx.getSystemInfo（OBJECT）：异步获取系统信息。getSystemInfo 的 OBJECT 参数说明如表 4-59 所示。

表 4-59　　getSystemInfo 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	是	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

用户通过 success 返回参数可以知道手机的相关信息，具体说明如表 4-60 所示。

表 4-60　　success 返回参数说明

参数	说明
model	手机型号
pixelRatio	设备像素比
screenWidth	屏幕宽度
screenHeight	屏幕高度
windowWidth	可使用窗口宽度
windowHeight	可使用窗口高度
language	微信设置的语言
version	微信版本号
system	操作系统版本

续表

参数	说明
platform	客户端平台
SDKVersion	客户端基础库版本

异步获取系统信息，代码如下所示：

```
// 异步获取系统信息
  testButtonClick1: function () {

    wx.getSystemInfo({
      success: function (res) {

        console.log(res.model)//手机型号
        console.log(res.pixelRatio)//设备像素比

        console.log(res.screenWidth)//屏幕宽度
        console.log(res.screenHeight)//屏幕高度
        console.log(res.windowWidth)//可使用窗口宽度
        console.log(res.windowHeight)//可使用窗口高度
        console.log(res.language)//微信设置的语言
        console.log(res.version)//微信版本号
        console.log(res.system)//操作系统版本
        console.log(res.platform)//客户端平台
        console.log(res.SDKVersion)//客户端基础库版本
      },
      fail: function (res) {
        console.log("获取系统信息失败")
      },
      complete: function (res) {
        console.log("获取系统信息完成")
      }
    })
  },
```

2）wx.getSystemInfoSync()：同步获取系统信息。使用代码如下所示：

```
// 同步获取系统信息
  testButtonClick2: function () {
    try {
      var res = wx.getSystemInfoSync()
      console.log(res.model)//手机型号
      console.log(res.pixelRatio)//设备像素比

      console.log(res.screenWidth)//屏幕宽度
      console.log(res.screenHeight)//屏幕高度
      console.log(res.windowWidth)//可使用窗口宽度
      console.log(res.windowHeight)//可使用窗口高度
      console.log(res.language)//微信设置的语言
      console.log(res.version)//微信版本号
      console.log(res.system)//操作系统版本
      console.log(res.platform)//客户端平台
      console.log(res.SDKVersion)//客户端基础库版本
    } catch (e) {
```

```
      //捕捉到错误时需做处理
    }
  },
```

screenWidth、screenHeight、SDKVersion 从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持，低版本需做兼容处理。

所谓兼容：

小程序的功能在不断地增加，有些旧版本的微信客户端并不支持新功能，所以在使用这些新功能的时候需要做兼容判断处理。

微信开发文档在组件、API 等页面描述中带有各个功能所支持的版本号。如果发现一个 API 或组件支持的版本号较高，不是所有版本都支持，用户在使用时可以通过 wx.getSystemInfo 或者 wx.getSystemInfoSync 获取到小程序的基础库版本号。

➢ **接口的兼容方式**

对于新增的 API，可以用以下代码来判断是否支持：

```
//API 兼容判断
  testAPI: function () {
    // 如果支持 openBluetoothAdapter API
    if (wx.openBluetoothAdapter) {
      wx.openBluetoothAdapter()
    } else {
      //不支持的话，弹出提示
      wx.showModal({
        title: '提示',
        content: '当前微信版本过低，无法使用该功能，请升级到最新微信版本后重试。'
      })
    }
  },
```

➢ **参数的兼容方式**

对于 API 的参数或者返回值有新增的参数，可以用以下代码判断：

```
//index.js
 //获取版本号
const SDKVersion = wx.getSystemInfoSync().SDKVersion || '1.0.0'

//版本号转换
const [MAJOR, MINOR, PATCH] = SDKVersion.split('.').map(Number)

const canIUse = apiName => {

if (apiName === 'showModal.cancel') {
    return MAJOR >= 1 && MINOR >= 1
  }
  return true
}

//参数兼容
```

```
  testParameter: function () {
    wx.showModal({
      success: function (res) {
        //调用 canIUse
        if (canIUse('showModal.cancel')) {
          console.log(res.cancel)
        }
      }
    })
  },
```

➢ 组件的兼容方式

对于组件，新增的属性在旧版本上不会被处理，不过也不会报错。如果特殊场景需要对旧版本做一些降级处理，代码如下所示：

```
//index.js
Page({
  data: {
    canIUse: canIUse('button.open-type.contact')
  }
})

<!--index.wxml-->
<button wx:if="{{canIUse}}" open-type="contact"> 客服消息 </button>
<contact-button wx:else></contact-button>
```

3）wx.canIUse（String）：判断小程序的 API、回调、参数、组件等是否在当前版本可用。

String 参数说明：使用${API}、${method}、${param}、${options}或者${component}、${attribute}、${option}方式来调用，例如：

- ${API} 代表 API 名字。
- ${method} 代表调用方式，有效值为 return、success、object、callback。
- ${param} 代表参数或者返回值。
- ${options} 代表参数的可选值。
- ${component} 代表组件名字。
- ${attribute} 代表组件属性。
- ${option} 代表组件属性的可选值。

示例代码如下所示：

```
wx.canIUse('openBluetoothAdapter')
wx.canIUse('getSystemInfoSync.return.screenWidth')
wx.canIUse('getSystemInfo.success.screenWidth')
wx.canIUse('showToast.object.image')
wx.canIUse('onCompassChange.callback.direction')
wx.canIUse('request.object.method.GET')
wx.canIUse('contact-button') wx.canIUse('text.selectable')
wx.canIUse('button.open-type.contact')
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

4.6.2 网络状态

1）wx.getNetworkType（OBJECT）：获取网络类型。getNetworkType 的 OBJECT 参数说明如表 4-61 所示。

表 4-61 getNetworkType 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	是	接口调用成功的回调函数，返回网络类型 networkType
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
// 获取网络类型
  networkButtonClick1: function () {
    wx.getNetworkType({
      success: function (res) {
        // 返回网络类型，有效值：
        // wifi/2g/3g/4g/unknown(Android 下不常见的网络类型)/none(无网络)
        var networkType = res.networkType
        console.log(networkType)
      }
    })
  },
```

2）wx.onNetworkStatusChange（CALLBACK）：监听网络状态变化。从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持。onNetworkStatusChange 的 CALLBACK 参数说明如表 4-62 所示。

表 4-62 onNetworkStatusChange 的 CALLBACK 参数说明

参数	类型	说明
isConnected	Boolean	当前是否有网络连接
networkType	String	网络类型，有效值：wifi/2g/3g/4g/unknown（Android 下不常见的网络类型）/none（无网络）

使用代码如下所示：

```
// 监听网络状态变化
  networkButtonClick2: function () {
    wx.onNetworkStatusChange(function (res) {
      console.log(res.isConnected)
      console.log(res.networkType)
    })
  },
```

4.6.3 重力加速度

1）wx.onAccelerometerChange（CALLBACK）：监听加速度数据，频率 5 次/秒，接口

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

调用后会自动开始监听，可使用 wx.stopAccelerometer 停止监听。onAccelerometerChange 的 CALLBACK 参数如表 4-63 所示。

表 4-63　　onAccelerometerChange 的 CALLBACK 参数

参数	类型
x	Number
y	Number
z	Number

使用代码如下所示：

```
//监听加速度数据
  accelerometerClick1: function () {
    wx.onAccelerometerChange(function (res) {
      console.log(res.x)
      console.log(res.y)
      console.log(res.z)
    })
  },
```

2）wx.startAccelerometer（OBJECT）：开始监听加速度数据。从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持。startAccelerometer 的 OBJECT 参数说明如表 4-64 所示。

表 4-64　　startAccelerometer 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
//开始监听加速度数据
  accelerometerClick2: function () {
    wx.startAccelerometer({

      //成功
      success: function (res) {
      },
      //失败
      fail: function (res) {
      },
      //完毕
      complete: function (res) {
      },
    })
  },
```

3）wx.stopAccelerometer（OBJECT）：停止监听加速度数据。从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持。stopAccelerometer 的 OBJECT 参数说明如表 4-65 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 4-65 stopAccelerometer 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
//停止监听加速度数据
  accelerometerClick3: function () {
    wx.stopAccelerometer({

      //成功
      success: function (res) {
      },
      //失败
      fail: function (res) {
      },
      //完毕
      complete: function (res) {
      },
    })
  },
```

4.6.4 罗盘

1）wx.onCompassChange（CALLBACK）：监听罗盘数据，频率 5 次/秒，接口调用后会自动开始监听，可使用 wx.stopCompass 停止监听。onCompassChange 的 CALLBACK 参数说明如表 4-66 所示。

表 4-66 onCompassChange 的 CALLBACK 参数说明

参数	类型	说明
direction	Number	面对的方向度数

使用代码如下所示：

```
// 监听罗盘数据
  compassClick1: function () {
    wx.onCompassChange(function (res) {
      console.log(res.direction)
    })
  },
```

2）wx.startCompass（OBJECT）：开始监听罗盘数据。从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持。startCompass 的 OBJECT 参数说明如表 4-67 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 4-67　　startCompass 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
//开始监听罗盘数据
  compassClick2: function () {
    wx.startCompass({

      //成功
      success: function (res) {
      },
      //失败
      fail: function (res) {
      },
      //完毕
      complete: function (res) {
      },
    })
  },
```

3）wx.stopCompass（OBJECT）：停止监听罗盘数据。从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持。stopCompass 的 OBJECT 参数说明如表 4-68 所示。

表 4-68　　stopCompass 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
// 结束监听罗盘数据
  compassClick3: function () {
    wx.stopCompass({

      //成功
      success: function (res) {
      },
      //失败
      fail: function (res) {
      },
      //完毕
      complete: function (res) {
      },
    })
  },
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

4.6.5 拨打电话

wx.makePhoneCall（OBJECT）：输入 phoneNumber 开始拨打电话。makePhoneCall 的 OBJECT 参数说明如表 4-69 所示。

表 4-69 makePhoneCall 的 OBJECT 参数说明

参数	类型	必填	说明
phoneNumber	String	是	需要拨打的电话号码
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数

使用代码如下所示：

```
//拨打电话
  phoneButtonClick: function () {
    wx.makePhoneCall({

      phoneNumber: '185****5656',//演示电话
      //成功
      success: function (res) {
      },
      //失败
      fail: function (res) {
      }
    })
  },
```

4.6.6 扫码

wx.scanCode（OBJECT）：调出客户端扫码界面，扫码成功后返回对应的结果。scanCode 的 OBJECT 参数说明如表 4-70 所示。

表 4-70 scanCode 的 OBJECT 参数说明

参数	类型	必填	说明
onlyFromCamera	Boolean	否	是否只能从相机扫码，不允许从相册选择图片扫码
success	Function	否	接口调用成功的回调函数，返回内容详见返回参数说明
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-71 所示。

表 4-71 success 返回参数说明

参数	说明
result	所扫码的内容

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

续表

参数	说明
scanType	所扫码的类型
charSet	所扫码的字符集
path	当所扫的码为当前小程序的合法二维码时，会返回此字段，内容为二维码携带的 path

扫码使用代码如下所示：

```
// 扫码
  scanCodeClick: function () {

   onlyFromCamera: true, //设置 true 后，只允许从相机扫码，不能从相册扫码

    wx.scanCode({
      //成功
      success: function (res) {
        console.log(res.result) //内容
        console.log(res.scanType)//类型
        console.log(res.charSet)//字符集
        console.log(res.path)//二维码携带的 path
      },
      //失败
      fail: function (res) {
      },
      //完成
      complete: function (res) {
      }
    })
  },
```

4.6.7 剪贴板

1）wx.setClipboardData（OBJECT）：设置系统剪贴板的内容。从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持。setClipboardData 的 OBJECT 参数说明如表 4-72 所示。

表 4-72　　setClipboardData 的 OBJECT 参数说明

参数	类型	必填	说明
data	String	是	需要设置的内容
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
// 设置系统剪贴板的内容
  clipboardClick1: function () {
    wx.setClipboardData({
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        data:"1234",
        //成功
        success: function (res) {
          console.log("设置成功")
        },
        //失败
        fail: function (res) {
        },
        //完成
        complete: function (res) {
        }
      })
    },
```

2）wx.getClipboardData（OBJECT）：获取系统剪贴板的内容。从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持。getClipboardData 的 OBJECT 参数说明如表 4-73 所示。

表 4-73　　getClipboardData 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-74 所示。

表 4-74　　success 返回参数说明

参数	类型	说明
data	String	剪贴板的内容

使用代码如下所示：

```
// 获取系统剪贴板的内容
  clipboardClick2: function () {
    wx.getClipboardData({
      //成功
      success: function (res) {
        console.log(res.data) //内容
      },
      //失败
      fail: function (res) {
      },
      //完成
      complete: function (res) {
      }
    })
  },
```

4.6.8　蓝牙

蓝牙适配器接口，从基础库版本 1.1.0 和 iOS 微信客户端 6.5.6 版本开始支持，Android

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

客户端暂不支持。由于系统的问题，目前仅在 Mac 版的开发工具上支持蓝牙调试。

1）wx.openBluetoothAdapter（OBJECT）：初始化蓝牙适配器。openBluetoothAdapter 的 OBJECT 参数说明如表 4-75 所示。

表 4-75　　openBluetoothAdapter 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	是	成功则返回成功初始化蓝牙适配器信息
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
// 初始化蓝牙适配器
  bluetoothClick1: function () {
    wx.openBluetoothAdapter({
      success: function (res) {
        console.log("openBluetoothAdapter success" + res)
      },
      fail: function (res) {
        console.log("openBluetoothAdapter fail" + res)
      },
      complete: function (res) {
        console.log("openBluetoothAdapter complete" + res)
      }
    })
  }
```

2）wx.closeBluetoothAdapter（OBJECT）：关闭蓝牙适配器。调用该方法将断开所有已建立的连接并释放系统资源。closeBluetoothAdapter 的 OBJECT 参数说明如表 4-76 所示。

表 4-76　　closeBluetoothAdapter 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	是	成功则返回成功关闭蓝牙适配器信息
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
// 关闭蓝牙适配器
  bluetoothClick2: function () {
    wx.closeBluetoothAdapter({
      success: function (res) {
        console.log("closeBluetoothAdapter success" + res)
      },
      fail: function (res) {
        console.log("closeBluetoothAdapter fail" + res)
      },
      complete: function (res) {
        console.log("closeBluetoothAdapter complete" + res)
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
      }
    })
  },
```

3）wx.getBluetoothAdapterState（OBJECT）：获取本机蓝牙适配器状态。getBluetoothAdapterState 的 OBJECT 参数说明如表 4-77 所示。

表 4-77　　getBluetoothAdapterState 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	是	成功则返回本机蓝牙适配器状态
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-78 所示。

表 4-78　　success 返回参数说明

参数	类型	说明
discovering	Boolean	是否正在搜索设备
available	Boolean	蓝牙适配器是否可用
errMsg	String	成功：ok；错误：详细信息

使用代码如下所示：

```
// 获取本机蓝牙适配器状态
  bluetoothClick3: function () {
    wx.getBluetoothAdapterState({
      success: function (res) {

        //是否正在搜索设备
        console.log("getBluetoothAdapterState success" + res.discovering)

        //蓝牙适配器是否可用
        console.log("getBluetoothAdapterState success" + res.available)

        //成功: ok; 错误: 详细信息
        console.log("getBluetoothAdapterState success" + res.errMsg)
      },
      fail: function (res) {
        console.log("getBluetoothAdapterState fail" + res)
      },
      complete: function (res) {
        console.log("getBluetoothAdapterState complete" + res)
      }
    })
  },
```

4）wx.onBluetoothAdapterStateChange（CALLBACK）：监听蓝牙适配器状态变化事件。onBluetoothAdapterStateChange 的 CALLBACK 参数说明如表 4-79 所示。

表 4-79 onBluetoothAdapterStateChange 的 CALLBACK 参数说明

参数	类型	说明
available	Boolean	蓝牙适配器是否可用
discovering	Boolean	蓝牙适配器是否处于搜索状态

使用代码如下所示：

```
// 监听蓝牙适配器状态变化事件
  bluetoothClick4: function () {
    wx.onBluetoothAdapterStateChange(function (res) {
      console.log('adapterState changed, available=', available)
      console.log('adapterState changed, discovering=', res)
    })
  },
```

5）wx.startBluetoothDevicesDiscovery（OBJECT）：开始搜寻附近的蓝牙外围设备。该操作比较耗费系统资源，在搜索并连接到设备后需要调用 wx.stopBluetoothDevicesDiscovery 方法停止搜索。startBluetoothDevicesDiscovery 的 OBJECT 参数说明如表 4-80 所示。

表 4-80 startBluetoothDevicesDiscovery 的 OBJECT 参数说明

参数	类型	必填	说明
services	Array	否	蓝牙设备主 service 的 uuid 列表
success	Function	是	成功则返回本机蓝牙适配器状态
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

services 参数说明：某些蓝牙设备会广播自己的主 service 的 uuid。如果在这里输入该数组，wx.startBluetoothDevicesDiscovery 会根据该 uuid 列表，只搜索有这个主 service 的设备。以微信硬件平台的蓝牙智能灯为例，主 service 的 uuid 是 FEE7，输入 FEE7，wx.startBluetoothDevicesDiscovery 只搜索主 service uuid 为 FEE7 的设备。

success 返回参数说明如表 4-81 所示。

表 4-81 success 返回参数说明

参数	类型	说明
errMsg	String	成功：ok；错误：详细信息

使用代码如下所示：

```
// 开始搜寻附近的蓝牙外围设备
  bluetoothClick5: function () {
    wx.startBluetoothDevicesDiscovery({
      services: ['FEE7'],
      success: function (res) {
        console.log(res.errMsg)
      },
      fail: function (res) {
        console.log(res)
```

```
      },
      complete: function (res) {
        console.log(res)
      }
    })
  },
```

6）wx.stopBluetoothDevicesDiscovery（OBJECT）：停止搜寻附近的蓝牙外围设备。请在确保找到需要连接的设备后调用该方法停止搜索。stopBluetoothDevicesDiscovery 的 OBJECT 参数说明如表 4-82 所示。

表 4-82　　stopBluetoothDevicesDiscovery 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	是	成功则返回本机蓝牙适配器状态
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-83 所示。

表 4-83　　success 返回参数说明

参数	类型	说明
errMsg	String	成功：ok；错误：详细信息

使用代码如下所示：

```
// 停止搜寻附近的蓝牙外围设备
  bluetoothClick6: function () {
    wx.stopBluetoothDevicesDiscovery({
      success: function (res) {
        console.log(res.errMsg)
      },
      fail: function (res) {
        console.log(res)
      },
      complete: function (res) {
        console.log(res)
      }
    })
  },
```

7）wx.getBluetoothDevices（OBJECT）：获取所有已发现的蓝牙设备，包括已经和本机处于连接状态的设备。getBluetoothDevices 的 OBJECT 参数说明如表 4-84 所示。

表 4-84　　getBluetoothDevices 的 OBJECT 参数说明

参数	类型	必填	说明
services	Array	否	蓝牙设备主 service 的 uuid 列表
success	Function	是	成功则返回本机蓝牙适配器状态
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-85 所示。

表 4-85 success 返回参数说明

参数	类型	说明
devices	Array	uuid 对应的已连接设备列表
errMsg	String	成功：ok；错误：详细信息

devices 参数中，每个对象包含的蓝牙设备参数说明，如表 4-86 所示。

表 4-86 devices 参数中，每个对象包含的蓝牙设备参数说明

参数	类型	说明
name	String	蓝牙设备名称，某些设备可能没有
deviceId	String	用于区分设备的 id
RSSI	Int	当前蓝牙设备的信号强度
advertisData	ArrayBuffer	当前蓝牙设备的广播内容

MacOS 系统可能无法获取 advertisData 及 RSSI，请使用真机调试，开发者工具和 Android 系统上获取到的 deviceId 为设备 MAC 地址，iOS 上则为设备的 uuid，因此 deviceId 不能硬编码到代码中。

使用代码如下所示：

```
// 获取所有已发现的蓝牙设备
  bluetoothClick7: function () {
    wx.getBluetoothDevices({
      // services: ['FEE7'],
      success: function (res) {
        console.log(res.devices)
        console.log(res.errMsg)

        for (var obj in res.devices) {

          console.log(res.devices[obj].name)
          console.log(res.devices[obj].deviceId)
          console.log(res.devices[obj].RSSI)
          console.log(res.devices[obj].advertisData)
        }
      },
      fail: function (res) {
        console.log(res)
      },
      complete: function (res) {
        console.log(res)
      }
    })
  },
```

8）wx.onBluetoothDeviceFound（CALLBACK）：监听寻找到新设备的事件。onBluetoothDeviceFound 的 CALLBACK 参数说明如表 4-87 所示。

表 4-87 onBluetoothDeviceFound 的 CALLBACK 参数说明

参数	类型	说明
devices	Array	新搜索到的设备列表

devices 参数中每一个对象包含的蓝牙设备信息参数说明，如表 4-88 所示。

表 4-88 devices 参数中，每个对象包含的蓝牙设备参数说明

参数	类型	说明
deviceId	String	蓝牙设备 id，参考 device 对象
name	String	蓝牙设备名称，参考 device 对象
RSSI	Int	当前蓝牙设备的信号强度
advertisData	ArrayBuffer	当前蓝牙设备的广播内容

MacOS 系统可能无法获取 advertisData 及 RSSI，请使用真机调试，开发者工具和 Android 系统上获取到的 deviceId 为设备 MAC 地址，iOS 上则为设备的 uuid，因此 deviceId 不能硬编码到代码中。

使用代码如下所示：

```
// 监听寻找到新设备的事件
  bluetoothClick8: function () {

    wx.onBluetoothDeviceFound(function (devices) {
      console.log('new device list has founded')
      console.dir(devices)
    })
  },
```

9）wx.getConnectedBluetoothDevices（OBJECT）：根据 uuid 获取处于已连接状态的设备。getConnectedBluetoothDevices 的 OBJECT 参数说明如表 4-89 所示。

表 4-89 getConnectedBluetoothDevices 的 OBJECT 参数说明

参数	类型	必填	说明
services	Array	是	蓝牙设备主 service 的 uuid 列表
success	Function	是	成功则返回本机蓝牙适配器状态
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-90 所示。

表 4-90 success 返回参数说明

参数	类型	说明
devices	Array	搜索到的设备列表
errMsg	String	成功：ok；错误：详细信息

devices 参数中每一个对象包含的蓝牙设备参数说明，如表 4-91 所示。

表 4-91　　devices 参数中，每个对象包含的蓝牙设备参数说明

参数	类型	说明
name	String	蓝牙设备名称，某些设备可能没有
deviceId	String	用于区分设备的 id

开发者工具和 Android 系统上获取到的 deviceId 为设备 MAC 地址，iOS 上则为设备的 uuid。因此 deviceId 不能硬编码到代码中。

使用代码如下所示：

```
// 根据 uuid 获取处于已连接状态的设备
  bluetoothClick9: function () {

    wx.getConnectedBluetoothDevices({
      success: function (res) {
        console.log(res)
      }
    })
  },
```

以下接口是低功耗蓝牙接口。

10）wx.createBLEConnection（OBJECT）：连接低功耗蓝牙设备。createBLEConnection 的 OBJECT 参数说明如表 4-92 所示。

表 4-92　　createBLEConnection 的 OBJECT 参数说明

参数	类型	必填	说明
deviceId	String	是	蓝牙设备 id，参考 getDevices 接口
success	Function	是	成功则返回本机蓝牙适配器状态
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-93 所示。

表 4-93　　success 返回参数说明

参数	类型	说明
errMsg	String	成功：ok；错误：详细信息

使用代码如下所示：

```
// 连接低功耗蓝牙设备
  bluetoothClick10: function () {

    wx.createBLEConnection({
      // 这里的 deviceId 需要从上面的 getBluetoothDevices 或 onBluetoothDeviceFound
      // 接口中获取
```

```
      deviceId: deviceId,
      success: function (res) {
        console.log(res)
      }
    })
  },
```

11）wx.closeBLEConnection（OBJECT）：断开与低功耗蓝牙设备的连接。closeBLEConnection 的 OBJECT 参数说明如表 4-94 所示。

表 4-94　　closeBLEConnection 的 OBJECT 参数说明

参数	类型	必填	说明
deviceId	String	是	蓝牙设备 id，参考 getDevices 接口
success	Function	是	成功则返回本机蓝牙适配器状态
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-95 所示。

表 4-95　　success 返回参数说明

参数	类型	说明
errMsg	String	成功：ok；错误：详细信息

使用代码如下所示：

```
// 断开与低功耗蓝牙设备的连接
  bluetoothClick11: function () {

    wx.closeBLEConnection({
      success: function (res) {
        console.log(res)
      }
    })
  },
```

12）wx.onBLEConnectionStateChanged（CALLBACK）：监听低功耗蓝牙连接的错误事件，包括设备丢失、连接异常断开等。onBLEConnectionStateChanged 的 CALLBACK 参数说明如表 4-96 所示。

表 4-96　　onBLEConnectionStateChanged 的 CALLBACK 参数说明

参数	类型	说明
deviceId	String	蓝牙设备 id，参考 device 对象
connected	Boolean	连接目前的状态

使用代码如下所示：

```
// 监听低功耗蓝牙连接的错误事件
  bluetoothClick12: function () {
```

```
        wx.onBLEConnectionStateChanged(function (res) {
          console.log('device ${res.deviceId} state has changed, connected:
${res.connected}')
        })
      },
```

13）wx.getBLEDeviceServices（OBJECT）：获取蓝牙设备所有 service（服务）。getBLEDeviceServices 的 OBJECT 参数说明如表 4-97 所示。

表 4-97　　getBLEDeviceServices 的 OBJECT 参数说明

参数	类型	必填	说明
deviceId	String	是	蓝牙设备 id，参考 getDevices 接口
success	Function	是	成功则返回本机蓝牙适配器状态
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-98 所示。

表 4-98　　success 返回参数说明

参数	类型	说明
services	Array	设备服务列表
errMsg	String	成功：ok；错误：详细信息

services 参数中每一个对象包含的蓝牙设备 service（服务）参数说明，如表 4-99 所示。

表 4-99　services 参数中，每个对象包含的蓝牙设备 service（服务）参数说明

参数	类型	说明
uuid	String	蓝牙设备服务的 uuid
isPrimary	Boolean	该服务是否为主服务

使用代码如下所示：

```
    // 获取蓝牙设备所有 service(服务)
    bluetoothClick13: function () {

      wx.getBLEDeviceServices({
        // 这里的 deviceId 需要从上面的 getBluetoothDevices 或 onBluetoothDeviceFound
        // 接口中获取
        deviceId: deviceId,
        success: function (res) {
          console.log('device services:', res.services)
        }
      })
    },
```

14）wx.getBLEDeviceCharacteristics（OBJECT）：获取蓝牙设备所有 characteristic（特征值）。getBLEDeviceCharacteristics 的 OBJECT 参数说明如表 4-100 所示。

表 4-100 getBLEDeviceCharacteristics 的 OBJECT 参数说明

参数	类型	必填	说明
deviceId	String	是	蓝牙设备 id，参考 device 对象
serviceId	String	是	蓝牙服务 uuid
success	Function	是	成功则返回本机蓝牙适配器状态
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-101 所示。

表 4-101 success 返回参数说明

参数	类型	说明
characteristics	Array	设备特征值列表
errMsg	String	成功：ok；错误：详细信息

characteristics 参数中，每个对象包含的蓝牙设备 characteristic（特征值）参数说明，如表 4-102 所示。

表 4-102 characteristics 参数中，每个对象包含的蓝牙设备 characteristic（特征值）参数说明

参数	类型	说明
uuid	String	蓝牙设备特征值的 uuid
properties	Object	该特征值支持的操作类型

properties 参数中每个对象包含的特征值参数说明，如表 4-103 所示。

表 4-103 properties 参数中每个对象包含的特征值参数说明

参数	类型	说明
read	Boolean	该特征值是否支持 read 操作
write	Boolean	该特征值是否支持 write 操作
notify	Boolean	该特征值是否支持 notify 操作
indicate	Boolean	该特征值是否支持 indicate 操作

使用代码如下所示：

```
// 获取蓝牙设备所有 characteristic(特征值)
  bluetoothClick14: function () {

    wx.getBLEDeviceCharacteristics({
      // 这里的 deviceId 需要从上面的 getBluetoothDevices 或 onBluetoothDeviceFound
      // 接口中获取
      deviceId: deviceId,
      // 这里的 serviceId 需要从上面的 getBLEDeviceServices 接口中获取
      serviceId: serviceId,
      success: function (res) {
```

```
        console.log('device getBLEDeviceCharacteristics:', res.characteristics)
      }
    })
  },
```

15）wx.readBLECharacteristicValue（OBJECT）：读取低功耗蓝牙设备特征值的二进制数据值。注意：设备的特征值必须支持 read 才可以成功调用，具体参照 characteristic 的 properties 属性。readBLECharacteristicValue 的 OBJECT 参数说明如表 4-104 所示。

表 4-104　　readBLECharacteristicValue 的 OBJECT 参数说明

参数	类型	必填	说明
deviceId	String	是	蓝牙设备 id，参考 device 对象
serviceId	String	是	蓝牙特征值对应服务的 uuid
characteristicId	String	是	蓝牙特征值的 uuid
success	Function	是	成功则返回本机蓝牙适配器状态
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-105 所示。

表 4-105　　success 返回参数说明

参数	类型	说明
characteristic	Object	设备特征值信息
errMsg	String	成功：ok；错误：详细信息

characteristics 参数中，每个对象包含的蓝牙设备 characteristic（特征值）参数说明，如表 4-106 所示。

表 4-106　　characteristics 参数中，每个对象包含的蓝牙设备 characteristic（特征值）参数说明

参数	类型	说明
characteristicId	String	蓝牙设备特征值的 uuid
serviceId	Object	蓝牙设备特征值对应服务的 uuid
value	ArrayBuffer	蓝牙设备特征值对应的二进制值

并行调用多次读写接口存在读写失败的可能性。read 接口读取到的信息需要从 onBLECharacteristicValueChange 方法的回调中获取。

使用代码如下所示：

```
// 读取低功耗蓝牙设备特征值的二进制数据值
  bluetoothClick15: function () {

    // 必须从这里的回调中才能获取
    wx.onBLECharacteristicValueChange(function (characteristic) {
      console.log('characteristic value comed:', characteristic)
```

```
    })

    wx.readBLECharacteristicValue({
      // 这里的 deviceId 需要从上面的 getBluetoothDevices 或 onBluetoothDeviceFound
      // 接口中获取
      deviceId: deviceId,
      // 这里的 serviceId 需要从上面的 getBLEDeviceServices 接口中获取
      serviceId: serviceId,
      // 这里的 characteristicId 需要从上面的 getBLEDeviceCharacteristics 接口中获取
      characteristicId: characteristicId,
      success: function (res) {
        console.log('readBLECharacteristicValue:', res.characteristic.value)
      }
    })
  },
```

16）wx.writeBLECharacteristicValue（OBJECT）：向低功耗蓝牙设备特征值中写入二进制数据。对应设备的特征值需要支持 write 才可以成功调用，具体参照 characteristic 的 properties 属性。并行调用多次读写接口存在读写失败的可能性。writeBLECharacteristicValue 的 OBJECT 参数说明如表 4-107 所示。

表 4-107　　writeBLECharacteristicValue 的 OBJECT 参数说明

参数	类型	必填	说明
deviceId	String	是	蓝牙设备 id，参考 device 对象
serviceId	String	是	蓝牙特征值对应服务的 uuid
characteristicId	String	是	蓝牙特征值的 uuid
value	ArrayBuffer	是	蓝牙设备特征值对应的二进制值
success	Function	是	成功则返回本机蓝牙适配器状态
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-108 所示。

表 4-108　　success 返回参数说明

参数	类型	说明
errMsg	String	成功：ok；错误：详细信息

使用代码如下所示：

```
// 向低功耗蓝牙设备特征值中写入二进制数据
  bluetoothClick16: function () {

    // 从这里的回调中可以获取到 write 导致的特征值改变
    wx.onBLECharacteristicValueChange(function (characteristic) {
      console.log('characteristic value changed:', characteristic)
    })

    // 向蓝牙设备发送一个 0x00 的十六进制数据
```

```
    let buffer = new ArrayBuffer(1)
    let dataView = new DataView(buffer)
    dataView.setUint8(0, 0)

    wx.writeBLECharacteristicValue({
      // 这里的 deviceId 需要从上面的 getBluetoothDevices 或 onBluetoothDeviceFound
      // 接口中获取
      deviceId: deviceId,
      // 这里的 serviceId 需要从上面的 getBLEDeviceServices 接口中获取
      serviceId: serviceId,
      // 这里的 characteristicId 需要从上面的 getBLEDeviceCharacteristics 接口中获取
      characteristicId: characteristicId,
      // 这里的 value 是 ArrayBuffer 类型
      value: buffer,
      success: function (res) {
        console.log('writeBLECharacteristicValue success', res.errMsg)
      }
    })
  },
```

17）wx.notifyBLECharacteristicValueChanged（OBJECT）：启用低功耗蓝牙设备特征值变化时的 notify 功能。设备的特征值必须支持 notify 才可以成功调用，具体参照 characteristic 的 properties 属性。另外，必须先启用 notify 才能监听到设备 characteristicValueChange 事件。notifyBLECharacteristicValueChanged 的 OBJECT 参数说明如表 4-109 所示。

表 4-109　notifyBLECharacteristicValueChanged 的 OBJECT 参数说明

参数	类型	必填	说明
deviceId	String	是	蓝牙设备 id，参考 device 对象
serviceId	String	是	蓝牙特征值对应服务的 uuid
characteristicId	String	是	蓝牙特征值的 uuid
state	Boolean	是	true：启用 notify；false：停用 notify
success	Function	是	成功则返回本机蓝牙适配器状态
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-110 所示。

表 4-110　success 返回参数说明

参数	类型	说明
errMsg	String	成功：ok；错误：详细信息

使用代码如下所示：

```
// 启用低功耗蓝牙设备特征值变化时的 notify 功能
  bluetoothClick17: function () {

    wx.notifyBLECharacteristicValueChanged({
      state: true, // 启用 notify 功能
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        // 这里的 deviceId 需要从上面的 getBluetoothDevices 或 onBluetoothDeviceFound
        // 接口中获取
        deviceId: deviceId,
        // 这里的 serviceId 需要从上面的 getBLEDeviceServices 接口中获取
        serviceId: serviceId,
        // 这里的 characteristicId 需要从上面的 getBLEDeviceCharacteristics 接口中获取
        characteristicId: characteristicId,
        success: function (res) {
          console.log('notifyBLECharacteristicValueChanged success', res.errMsg)
        }
      })
    },
```

18）wx.onBLECharacteristicValueChange（CALLBACK）：监听低功耗蓝牙设备的特征值变化。必须先启用 notify 接口才能接收到设备推送的 notification。onBLECharacteristicValueChange 的 CALLBACK 参数说明如表 4-111 所示。

表 4-111　　onBLECharacteristicValueChange 的 CALLBACK 参数说明

参数	类型	说明
deviceId	String	蓝牙设备 id，参考 device 对象
serviceId	String	特征值所属服务的 uuid
characteristicId	String	特征值的 uuid
value	ArrayBuffer	特征值最新的值

使用代码如下所示：

```
  // 监听低功耗蓝牙设备的特征值变化
    bluetoothClick18: function () {

      wx.onBLECharacteristicValueChange(function (res) {
        console.log('characteristic ${res.characteristicId} has changed, now
is ${res.value}')
      })
    },
```

蓝牙错误码（errCode）列表，如表 4-112 所示。

表 4-112　　蓝牙错误码

错误码	说明	备注
0	ok	正常
10000	not init	未初始化蓝牙适配器
10001	not available	当前蓝牙适配器不可用
10002	no device	没有找到指定设备
10003	connection fail	连接失败
10004	no service	没有找到指定服务
10005	no characteristic	没有找到指定特征值
10006	no connection	当前连接已断开

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

续表

错误码	说明	备注
10007	property not support	当前特征值不支持此操作
10008	system error	其余所有系统上报的异常
10009	system not support	Android 系统特有，系统版本低于 4.3 不支持 BLE

4.6.9 iBeacon

微信小程序从基础库 1.2.0 开始支持 iBeacon 功能。

1）wx.startBeaconDiscovery(OBJECT)：开始搜索附近的 iBeacon 设备。

startBeaconDiscovery 的 OBJECT 参数说明如表 4-113 所示。

表 4-113 startBeaconDiscovery 的 OBJECT 参数说明

参数	类型	必填	说明
uuids	StringArray	是	iBeacon 设备广播的 uuids
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-114 所示。

表 4-114 success 返回参数说明

参数	类型	说明
errMsg	String	调用结果

使用代码如下所示：

```
//开始搜索附近的 iBeacon 设备
  bluetoothClick21: function () {
    wx.startBeaconDiscovery({
      success(res) {
      }
    })
  },
```

2）wx. stopBeaconDiscovery(OBJECT)：停止搜索附近的 iBeacon 设备。

stopBeaconDiscovery 的 OBJECT 参数说明如表 4-115 所示。

表 4-115 stopBeaconDiscovery 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

success 返回参数说明如表 4-116 所示。

表 4-116 success 返回参数说明

参数	类型	说明
errMsg	String	调用结果

使用代码如下所示：

```
//停止搜索附近的 iBeacon 设备
  bluetoothClick22: function () {
    wx.stopBeaconDiscovery({
      success(res) {
      }
    })
  },
```

3）wx. getBeacons(OBJECT)：获取所有已搜索到的 iBeacon 设备。

getBeacons 的 OBJECT 参数说明如表 4-117 所示。

表 4-117 getBeacons 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-118 所示。

表 4-118 success 返回参数说明

参数	类型	说明
Beacons	ObjectArray	iBeacon 设备列表
errMsg	String	调用结果

iBeacon 结构如表 4-119 所示。

表 4-119 iBeacon 结构

参数	类型	说明
uuid	String	iBeacon 设备广播的 uuid
major	String	iBeacon 设备的主 id
minor	String	iBeacon 设备的次 id
proximity	Number	表示设备距离的枚举值
accuracy	Number	iBeacon 设备的距离
rssi	Number	表示设备的信号强度

使用代码如下所示：

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
//获取所有已搜索到的 iBeacon 设备
  bluetoothClick23: function () {
    wx.getBeacons({
      success(res) {
      }
    })
  },
```

4）wx.onBeaconUpdate(CALLBACK)：监听 iBeacon 设备的更新事件。

onBeaconUpdate 的 CALLBACK 参数说明如表 4-120 所示。

表 4-120　　onBeaconUpdate 的 CALLBACK 参数说明

参数	类型	说明
beacons	Array Object	当前搜寻到的所有 iBeacon 设备列表

iBeacon 结构如表 4-121 所示。

表 4-121　　iBeacon 结构

参数	类型	说明
uuid	String	iBeacon 设备广播的 uuid
major	String	iBeacon 设备的主 id
minor	String	iBeacon 设备的次 id
proximity	Number	表示设备距离的枚举值
accuracy	Number	iBeacon 设备的距离
rssi	Number	表示设备的信号强度

使用代码如下所示：

```
//监听 iBeacon 设备的更新事件
  bluetoothClick24: function () {
    wx.onBeaconUpdate(function(res){

    })
  },
```

5）wx.onBeaconServiceChange(CALLBACK)：监听 iBeacon 服务的状态变化。

onBeaconServiceChange 的 CALLBACK 参数说明如表 4-122 所示。

表 4-122　　onBeaconServiceChange 的 CALLBACK 参数说明

参数	类型	说明
available	Boolean	服务目前是否可用
discovering	Boolean	目前是否处于搜索状态

关于 iBeacon 的错误码列表，如表 4-123 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 4-123 iBeacon 错误码列表

错误码	说明	备注
0	ok	正常
11000	unsupport	系统或设备不支持
11001	bluetooth service unavailable	蓝牙服务不可用
11002	location service unavailable	位置服务不可用
11003	already start	已经开始搜索

使用代码如下所示：

```
//监听 iBeacon 服务的状态变化
  bluetoothClick25: function () {
    wx.onBeaconServiceChange(function (res) {
    })
  },
```

4.6.10 屏幕亮度

微信小程序从基础库 1.2.0 开始支持获取或设置屏幕亮度。

1）wx.setScreenBrightness(OBJECT)：设置屏幕亮度。

setScreenBrightness 的 OBJECT 参数说明如表 4-124 所示。

表 4-124 setScreenBrightness 的 OBJECT 参数说明

参数	类型	必填	说明
value	Number	是	屏幕亮度值，范围 0～1，0 最暗，1 最亮
success	Function	否	接口调用成功的回调参数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
//设置屏幕亮度
  bluetoothClick31: function () {
    wx.setScreenBrightness({
      value: 0.5,
    })
  },
```

2）wx.getScreenBrightness(OBJECT)：获取屏幕亮度。

getScreenBrightness 的 OBJECT 参数说明如表 4-125 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 4-125 getScreenBrightness 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-126 所示。

表 4-126 success 返回参数说明

参数	类型	说明
value	Number	屏幕亮度值，范围 0～1，0 最暗，1 最亮

使用代码如下所示：

```
//获取屏幕亮度
  bluetoothClick32: function () {
    wx.getScreenBrightness({
      success(res) {
      }
    })
  },
```

3）wx.setKeepScreenOn(OBJECT)：设置是否保持常亮状态。仅在当前小程序生效，离开小程序后设置失效。（该接口从基础库 1.4.0 开始支持）

setKeepScreenOn 的 OBJECT 参数说明如表 4-127 所示。

表 4-127 setKeepScreenOn 的 OBJECT 参数说明

参数	类型	必填	说明
keepScreenOn	Boolean	是	是否保持屏幕常亮
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-128 所示。

表 4-128 success 返回参数说明

参数	类型	说明
errMsg	String	调用结果

使用代码如下所示：

```
//设置是否保持常亮状态
  bluetoothClick33: function () {
    wx.setKeepScreenOn({
      keepScreenOn: true,
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    })
  },
```

4.6.11 用户截屏事件

微信小程序从基础库 1.4.0 开始支持 wx.onUserCaptureScreen(CALLBACK)，监听用户截屏事件，用户使用系统截屏按键截屏时触发此事件。

使用代码如下所示：

```
//用户截屏事件
  bluetoothClick41: function () {
    wx.onUserCaptureScreen(function (res) {
      console.log('用户截屏了')
    })
  },
```

4.6.12 振动

微信小程序从基础库 1.2.0 开始支持手机发生振动。

1）wx.vibrateLong(OBJECT)：使手机发生较长时间的振动（400ms）。

vibrateLong 的 OBJECT 参数说明如表 4-129 所示。

表 4-129　　vibrateLong 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
//使手机发生较长时间的振动（400ms）
  bluetoothClick51: function () {
    wx.vibrateLong({
      success(res) {
      },
      fail(res) {
      },
      complete(res) {
      }
    })
  },
```

2）wx.vibrateShort(OBJECT)：使手机发生较短时间的振动（15ms）。

vibrateShort 的 OBJECT 参数说明如表 4-130 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 4-130 vibrateShort 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
//使手机发生较短时间的振动（15ms）
  bluetoothClick52: function () {
    wx.vibrateShort({
      success(res) {
      },
      fail(res) {
      },
      complete(res) {
      }
    })
  },
```

4.6.13 手机联系人

微信小程序从基础库 1.2.0 开始支持 wx.addPhoneContact(OBJECT)，调用后，用户可以选择将该表单以“新建联系人”或“添加到已有联系人”的方式写入手机系统通讯录，完成手机通讯录联系人和联系方式的添加。

addPhoneContact 的 OBJECT 参数说明如表 4-131 所示。

表 4-131 addPhoneContact 的 OBJECT 参数说明

参数	类型	必填	说明
photoFilePath	String	否	头像本地文件路径
nickName	String	否	昵称
lastName	String	否	姓氏
middleName	String	否	中间名
firstName	String	是	名字
remark	String	否	备注
mobilePhoneNumber	String	否	手机号
weChatNumber	String	否	微信号
addressCountry	String	否	联系地址所属国家
addressState	String	否	联系地址所属省份
addressCity	String	否	联系地址所属城市
addressStreet	String	否	联系地址所属街道
addressPostalCode	String	否	联系地址邮政编码

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

续表

参数	类型	必填	说明
organization	String	否	公司
title	String	否	职位
workFaxNumber	String	否	工作传真
workPhoneNumber	String	否	工作电话
hostNumber	String	否	公司电话
email	String	否	电子邮件
url	String	否	网站
workAddressCountry	String	否	工作地址所属国家
workAddressState	String	否	工作地址所属省份
workAddressCity	String	否	工作地址所属城市
workAddressStreet	String	否	工作地址所属街道
workAddressPostalCode	String	否	工作地址邮政编码
homeFaxNumber	String	否	住宅传真
homePhoneNumber	String	否	住宅电话
homeAddressCountry	String	否	住宅地址所属国家
homeAddressState	String	否	住宅地址所属省份
homeAddressCity	String	否	住宅地址所属城市
homeAddressStreet	String	否	住宅地址所属街道
homeAddressPostalCode	String	否	住宅地址邮政编码
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

addPhoneContact 的回调参数，如表 4-132 所示。

表 4-132　　addPhoneContact 的回调参数

回调类型	errMsg	说明
success	ok	添加成功
fail	fail cancel	用户取消操作
fail	fail ${detail}	调用失败，detail 为详细信息

使用代码如下所示：

```
//添加联系人
  bluetoothClick61: function () {

    wx.addPhoneContact({
      firstName: '刘明洋',
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
      mobilePhoneNumber: '18500000000'
    })
  }
```

4.6.14 NFC

此功能暂仅支持 HCE（基于主机的卡模拟）模式，即将安卓手机模拟成实体智能卡。

适用机型：支持 NFC 功能，且系统版本为 Android 5.0 及以上的手机。

适用卡范围：符合 ISO 14443-4 标准的 CPU 卡。

1）wx.getHCEState(OBJECT)：判断当前设备是否支持 HCE 模式。从基础库 1.7.0 开始支持，低版本需做兼容处理。

getHCEState 的 OBJECT 参数说明如表 4-133 所示。

表 4-133 getHCEState 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

其中，success 返回参数说明如表 4-134 所示。

表 4-134 success 返回参数说明

参数	类型	说明
errMsg	String	错误信息
errCode	Number	错误码

示例代码如下所示：

```
wx.getHCEState({
  success: function(res) {
    console.log(res.errCode)
  }
})
```

2）wx.startHCE(OBJECT)：初始化 NFC 模块。从基础库 1.7.0 开始支持，低版本需做兼容处理。

startHCE 的 OBJECT 参数说明如表 4-135 所示。

表 4-135 startHCE 的 OBJECT 参数说明

参数	类型	必填	说明
aid_list	Array	是	需要注册到系统的 AID 列表，每个 AID 为 String 类型
success	Function	否	接口调用成功的回调函数

非卖品，仅供非商业用途或交流学习使用

续表

参数	类型	必填	说明
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

其中，success 返回参数说明如表 4-136 所示。

表 4-136　　success 返回参数说明

参数	类型	说明
errMsg	String	错误信息
errCode	Number	错误码

示例代码如下所示：

```
wx.startHCE({
    aid_list: ['F222222222']
    success: function(res) {
        console.log(res.errMsg)
    }
})
```

3）wx.stopHCE(OBJECT)：关闭 NFC 模块。仅在安卓系统下有效。从基础库 1.7.0 开始支持，低版本需做兼容处理。

stopHCE 的 OBJECT 参数说明如表 4-137 所示。

表 4-137　　stopHCE 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

其中，success 返回参数说明如表 4-138 所示。

表 4-138　　success 返回参数说明

参数	类型	说明
errMsg	String	错误信息
errCode	Number	错误码

示例代码如下所示：

```
wx.stopHCE({
    success: function(res) {
        console.log(res.errMsg)
    }
})
```

4）wx.onHCEMessage(CALLBACK)：监听 NFC 设备的消息回调，并在回调中处理。从基础库 1.7.0 开始支持，低版本需做兼容处理。onHCEMessage 的 CALLBACK 参数中 messageType 表示消息类型，目前有如下值：

（1）消息为 HCE Apdu Command 类型，小程序需对此指令进行处理，并调用 sendHCEMessage 接口返回处理指令；

（2）消息为设备离场事件。

onHCEMessage 的 CALLBACK 参数说明如表 4-139 所示。

表 4-139　　onHCEMessage 的 CALLBACK 参数说明

参数	类型	说明
messageType	Number	消息类型
data	ArrayBuffer	客户端接收到 NFC 设备的指令，此参数仅当 messageType=1 时有效
reason	Number	此参数仅当 messageType=2 时有效

5）wx.sendHCEMessage(OBJECT)：发送 NFC 消息。仅在安卓系统下有效。从基础库 1.7.0 开始支持，低版本需做兼容处理。

sendHCEMessage 的 OBJECT 参数说明如表 4-140 所示。

表 4-140　　sendHCEMessage 的 OBJECT 参数说明

参数	类型	必填	说明
data	ArrayBuffer	是	二进制数据
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

其中，success 返回参数说明如表 4-141 所示。

表 4-141　　success 返回参数说明

参数	类型	说明
errMsg	String	错误信息
errCode	Number	错误码

示例代码如下所示：

```
const buffer = new ArrayBuffer(1)
const dataView = new DataView(buffer)
dataView.setUint8(0, 0)

wx.startHCE({
```

```
success: function(res) {
 wx.onHCEMessage(function(res) {
if (res.messageType === 1) {
 wx.sendHCEMessage({data: buffer})
  }
  })
  }
})
```

每个接口调用的时候，都会返回 errCode 字段。errCode（错误码）列表如表 4-142 所示。

表 4-142　　errCode（错误码）列表

错误码	说明
0	ok
13000	当前设备不支持 NFC
13001	当前设备支持 NFC，但系统 NFC 开关未开启
13002	当前设备支持 NFC，但不支持 HCE
13003	AID 列表参数格式错误
13004	未设置微信为默认 NFC 支付应用
13005	返回的指令不合法
13006	注册 AID 失败

4.6.15 Wi-Fi

小程序支持搜索周边的 Wi-Fi。用户可以选择一个 Wi-Fi，输入密码发起连接。该系列接口为系统原生接口。

1. Android 和 iOS 系统（仅支持 iOS 11 及以上版本）下连接指定 Wi-Fi 接口调用时序：

startWifi→connectWifi→onWifiConnected

2. 连接周边 Wi-Fi 接口调用时序：

在 Android 系统下：

startWifi→getWifiList→onGetWifiList→connectWifi→onWifiConnected

在 iOS 系统下（iOS 11.0 及 11.1 版本因系统原因暂不支持）：

startWifi→getWifiList→onGetWifiList→setWifiList→onWifiConnected

Wi-Fi 相关接口暂不可用 wx.canIUse 接口判断。Android 6.0 以上版本，在没有打开定位开关时会导致设备不能正常获取周边的 Wi-Fi 信息。

1）wx.startWifi(OBJECT)：初始化 Wi-Fi 模块。从基础库 1.6.0 开始支持，低版本需做兼容处理。

startWifi 的 OBJECT 参数说明如表 4-143 所示。

表 4-143 startWifi 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码如下所示：

```
wx.startWifi({
success: function(res) {
 console.log(res.errMsg)
 }
})
```

2）wx.stopWifi(OBJECT)：关闭 Wi-Fi 模块。从基础库 1.6.0 开始支持，低版本需做兼容处理。

stopWifi 的 OBJECT 参数说明如表 4-144 所示。

表 4-144 stopWifi 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码如下所示：

```
wx.stopWifi({
success: function(res) {
console.log(res.errMsg)
 }
})
```

3）wx.connectWifi(OBJECT)：连接 Wi-Fi。若已知 Wi-Fi 信息，可以直接利用该接口连接。仅 Android 与 iOS 11 以上版本支持。从基础库 1.6.0 开始支持，低版本需做兼容处理。

connectWifi 的 OBJECT 参数说明如表 4-145 所示。

表 4-145 connectWifi 的 OBJECT 参数说明

参数	类型	必填
SSID	String	是
BSSID	String	是
password	String	否

续表

参数	类型	必填
success	Function	否
fail	Function	否
complete	Function	否

示例代码如下所示：

```
wx.connectWifi({
  SSID: '',
 BSSID: '',
success: function(res) {
 console.log(res.errMsg)
}
})
```

4）wx.getWifiList(OBJECT)：请求获取 Wi-Fi 列表，在 onGetWifiList 注册的回调中返回 wifiList 数据。iOS 将跳转到系统的 Wi-Fi 界面，Android 不会跳转。iOS 11.0 及 iOS 11.1 两个版本因系统问题，该方法失效，但在 iOS 11.2 中已修复。从基础库 1.6.0 开始支持，低版本需做兼容处理。

getWifiList 的 OBJECT 参数说明如表 4-146 所示。

表 4-146　　getWifiList 的 OBJECT 参数说明

参数	类型	必填
success	Function	否
fail	Function	否
complete	Function	否

5）wx.onGetWifiList(CALLBACK)：监听在获取到 Wi-Fi 列表数据时的事件，在回调中将返回 wifiList。从基础库 1.6.0 开始支持，低版本需做兼容处理。

onGetWifiList 的 CALLBACK 参数说明如表 4-147 所示。

表 4-147　　onGetWifiList 的 CALLBACK 参数说明

参数	类型	说明
wifiList	Array	Wi-Fi 列表数据

Wi-Fi 列表项说明如表 4-148 所示。

表 4-148　　Wi-Fi 列表项说明

参数	类型	说明
SSID	String	Wi-Fi 的 SSID

续表

参数	类型	说明
BSSID	String	Wi-Fi 的 BSSID
secure	Boolean	Wi-Fi 是否安全
signalStrength	Number	Wi-Fi 信号强度

6）wx.setWifiList(OBJECT)：iOS 特有接口，在 onGetWifiList 回调后，利用接口设置 wifiList 中 AP 的相关信息。从基础库 1.6.0 开始支持，低版本需做兼容处理。

使用该接口时的注意事项：

（1）该接口只能在 onGetWifiList 回调之后才能调用。

（2）此时客户端会被挂起，等待小程序设置 Wi-Fi 信息。请务必尽快调用该接口，若无数据请输入一个空数组。

（3）有可能随着周边 Wi-Fi 列表的刷新，单个流程内收到多次带有重复的 Wi-Fi 列表的回调。

setWifiList 的 OBJECT 参数说明如表 4-149 所示。

表 4-149　　setWifiList 的 OBJECT 参数说明

参数	类型	必填	说明
wifiList	Array	是	提供预设的 Wi-Fi 信息列表
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

Wi-Fi 信息列表项说明如表 4-150 所示。

表 4-150　　Wi-Fi 信息列表项说明

参数	类型	说明
SSID	String	Wi-Fi 的 SSID
BSSID	String	Wi-Fi 的 BSSID
password	String	Wi-Fi 的密码

示例代码如下所示：

```
wx.onGetWifiList(function(res){
if (res.wifiList.length){
    wx.setWifiList({
      wifiList: [
          {
        SSID: res.wifiList[
              0
            ].SSID,
```

```
            BSSID: res.wifiList[
                  0
                ].BSSID,
            password: '123456'
              }
            ]
          })
        } else{
        wx.setWifiList({
          wifiList: []
          })
        }
    })
  wx.getWifiList()
```

7）wx.onWifiConnected(CALLBACK)：监听连接上 Wi-Fi 的事件。从基础库 1.6.0 开始支持，低版本需做兼容处理。

onWifiConnected 的 CALLBACK 参数说明如表 4-151 所示。

表 4-151　　onWifiConnected 的 CALLBACK 参数说明

参数	类型	说明
wifi	Object	Wi-Fi 信息

wifi 参数说明如表 4-152 所示。

表 4-152　　wifi 参数说明

参数	类型	说明
SSID	String	Wi-Fi 的 SSID
BSSID	String	Wi-Fi 的 BSSID
secure	Boolean	Wi-Fi 是否安全
signalStrength	Number	Wi-Fi 信号强度

8）wx.getConnectedWifi(OBJECT)：获取已连接的 Wi-Fi 信息。从基础库 1.6.0 开始支持，低版本需做兼容处理。

getConnectedWifi 的 OBJECT 参数说明如表 4-153 所示。

表 4-153　　getConnectedWifi 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-154 所示。

表 4-154 success 返回参数说明

参数	类型	说明
wifi	Object	Wi-Fi 信息

wifi 参数说明如表 4-155 所示。

表 4-155 wifi 参数说明

参数	类型	说明
SSID	String	Wi-Fi 的 SSID
BSSID	String	Wi-Fi 的 BSSID
secure	Boolean	Wi-Fi 是否安全
signalStrength	Number	Wi-Fi 信号强度

每个接口调用的时候，都会返回 errCode 字段。errCode（错误码）列表如表 4-156 所示。

表 4-156 errCode（错误码）列表

错误码	说明	备注
0	ok	正常
12000	not init	未先调用 startWifi 接口
12001	system not support	当前系统不支持相关功能
12002	password error	Wi-Fi 密码错误
12003	connection timeout	连接超时
12004	duplicate request	重复连接 Wi-Fi
12005	wifi not turned on	Android 特有，未打开 Wi-Fi 开关
12006	gps not turned on	Android 特有，未打开 GPS 定位开关
12007	user denied	用户拒绝授权连接 Wi-Fi
12008	invalid SSID	无效 SSID
12009	system config err	系统运营商配置拒绝连接 Wi-Fi
12010	system internal error	系统其他错误，需要在 errmsg 打印具体的错误原因
12011	weapp in background	应用在后台无法配置 Wi-Fi

4.7 界面交互

界面交互 API 包括交互反馈、设置导航条、页面导航、动画、绘图等功能。本节主要介绍前面几个功能。绘图功能将在 4.8 节讲解。

4.7.1 交互反馈

1）wx.showToast（OBJECT）：显示消息提示框。showToast 的 OBJECT 参数说明如

表 4-157 所示。

表 4-157　　showToast 的 OBJECT 参数说明

参数	类型	必填	说明
title	String	是	提示的内容
icon	String	否	图标，只支持 success、loading
image	String	否	自定义图标的本地路径，image 的优先级高于 icon
duration	Number	否	提示的延迟时间，单位为 ms，默认：1500
mask	Boolean	否	是否显示透明蒙层，防止触摸穿透，默认：false
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

image 字段从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持。具体使用代码如下所示：

```
//显示消息提示框
  actionButtonClick1: function () {
    wx.showToast({
      title: '成功',
      icon: 'success',
      duration: 2000
    })
  },
```

2）wx.showLoading（OBJECT）：显示 loading 提示框，需主动调用 wx.hideLoading 才能关闭提示框。从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持。showLoading 的 OBJECT 参数说明如表 4-158 所示。

表 4-158　　showLoading 的 OBJECT 参数说明

参数	类型	必填	说明
title	String	是	提示的内容
mask	Boolean	否	是否显示透明蒙层，防止触摸穿透，默认：false
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
// 显示 loading 提示框
  actionButtonClick2: function () {
    wx.showLoading({
      title: '加载中',
    })

    setTimeout(function () {
```

```
      wx.hideLoading()
    }, 2000)
  },
```

3）wx.hideToast()：隐藏消息提示框。

4）wx.hideLoading()：隐藏 loading 提示框，从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持。

代码如下所示：

```
// 隐藏 loading 提示框
  actionButtonClick3: function () {
    wx.showLoading({
      title: '加载中',
    })

    setTimeout(function () {
      wx.hideLoading()
    }, 2000)
  },
```

5）wx.showModal（OBJECT）：显示模态弹窗。showModal 的 OBJECT 参数说明如表 4-159 所示。

表 4-159　　showModal 的 OBJECT 参数说明

参数	类型	必填	说明
title	String	是	提示的标题
content	String	是	提示的内容
showCancel	Boolean	否	是否显示取消按钮，默认为 true
cancelText	String	否	取消按钮的文字，默认为"取消"，最多 4 个字符
cancelColor	HexColor	否	取消按钮的文字颜色，默认为"#000000"
confirmText	String	否	确定按钮的文字，默认为"确定"，最多 4 个字符
confirmColor	HexColor	否	确定按钮的文字颜色，默认为"#3CC51F"
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

succes 返回参数说明如表 4-160 所示。

表 4-160　　succes 返回参数说明

参数	类型	说明
confirm	Boolean	为 true 时，表示用户单击了确定按钮
cancel	Boolean	为 true 时，表示用户单击了取消按钮（用于 Android 系统区分单击蒙层关闭还是单击取消按钮关闭）

cancel 字段从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持。使用代码如下所示：

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
// 显示模态弹窗
  actionButtonClick4: function () {
    wx.showModal({
      title: '提示',
      content: '这是一个模态弹窗',
      success: function (res) {
        if (res.confirm) {
          console.log('用户单击确定')
        } else if (res.cancel) {
          console.log('用户单击取消')
        }
      }
    })
  },
```

6）wx.showActionSheet（OBJECT）：显示操作菜单。showActionSheet 的 OBJECT 参数说明如表 4-161 所示。

表 4-161　　showActionSheet 的 OBJECT 参数说明

参数	类型	必填	说明
itemList	String Array	是	按钮的文字数组，数组长度最大为 6 个
itemColor	HexColor	否	按钮的文字颜色，默认为"#000000"
success	Function	否	接口调用成功的回调函数，详见返回参数说明
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-162 所示。

表 4-162　　success 返回参数说明

参数	类型	说明
tapIndex	Number	用户单击的按钮，顺序从上到下，从 0 开始

使用代码如下所示：

```
// 显示操作菜单
  actionButtonClick5: function () {
    wx.showActionSheet({
      itemList: ['item1', 'item2', 'item3'],
      success: function (res) {
        console.log(res.tapIndex)
      },
      fail: function (res) {
        console.log(res.errMsg)
      }
    })
  },
```

4.7.2　导航和 tabBar 的使用介绍

页面导航可以设置导航条标题、显示或隐藏加载状态、实现页面的跳转。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

1）wx.setNavigationBarTitle（OBJECT）：动态设置当前页面的标题。setNavigationBarTitle 的 OBJECT 参数说明如表 4-163 所示。

表 4-163　　setNavigationBarTitle 的 OBJECT 参数说明

参数	类型	必填	说明
title	String	是	页面标题
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
// 动态设置当前页面的标题
  navButtonClick1: function () {
    wx.setNavigationBarTitle({
      title: '第四章 第 7 节',
      success: function (res) {
        console.log("setNavigationBarTitle success")
      },
      fail: function (res) {
        console.log("setNavigationBarTitle fail")
      },
      complete: function (res) {
        console.log("setNavigationBarTitle complete")
      }
    })
  },
```

2）wx.showNavigationBarLoading()：在当前页面显示导航条加载动画。

3）wx.hideNavigationBarLoading()：隐藏导航条加载动画。

4）wx.navigateTo（OBJECT）：保留当前页面，跳转到应用内的某个页面，使用 wx.navigateBack 可以返回到原页面。为了避免给用户在使用小程序时造成困扰，微信规定页面路径最多只能是 5 层，请尽量避免多层级的交互方式。navigateTo 的 OBJECT 参数说明如表 4-164 所示。

表 4-164　　navigateTo 的 OBJECT 参数说明

参数	类型	必填	说明
url	String	是	需要跳转的应用内非 tabBar 的页面的路径，路径后可以带参数。参数与路径之间使用?分隔，参数键与参数值用=相连，不同参数用&分隔，如'path?key=value&key2=value2'
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
// 保留当前页面，跳转到应用内的某个页面
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
  navButtonClick4: function () {
    wx.navigateTo({
      url: '../test1/test1?id=1',
      success: function (res) {
        console.log("navigateTo success")
      },
      fail: function (res) {
        console.log("navigateTo fail")
      },
      complete: function (res) {
        console.log("navigateTo complete")
      }
    })
  },

// pages/test1/test1.js
Page({
  data: {},
  onLoad: function (options) {
    // 页面初始化 options 为页面跳转所带来的参数
    console.log(options.id)
  }
})
```

5）wx.redirectTo（OBJECT）：关闭当前页面，跳转到应用内的某个页面，跳转之后不能返回。redirectTo 的 OBJECT 参数说明如表 4-165 所示。

表 4-165　　redirectTo 的 OBJECT 参数说明

参数	类型	必填	说明
url	String	是	需要跳转的应用内非 tabBar 的页面的路径，路径后可以带参数。参数与路径之间使用?分隔，参数键与参数值用=相连，不同参数用&分隔，如 'path?key=value&key2=value2'
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
// 关闭当前页面，跳转到应用内的某个页面
  navButtonClick5: function () {
    wx.redirectTo({
      url: '../test1/test1?id=1',
      success: function (res) {
        console.log("redirectTo success")
      },
      fail: function (res) {
        console.log("redirectTo fail")
      },
      complete: function (res) {
        console.log("redirectTo complete")
      }
    })
  },
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

6）wx.reLaunch（OBJECT）：关闭所有页面，打开到应用内的某个页面。从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持。reLaunch 的 OBJECT 参数说明如表 4-166 所示。

表 4-166　　reLaunch 的 OBJECT 参数说明

参数	类型	必填	说明
url	String	是	需要跳转的应用内非 tabBar 的页面的路径，路径后可以带参数。参数与路径之间使用?分隔，参数键与参数值用=相连，不同参数用&分隔，如 'path?key=value&key2=value2'
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
// 关闭所有页面，打开到应用内的某个页面
  navButtonClick6: function () {

    wx.reLaunch({
      url: 'pages/test1/test1',
      success: function (res) {
        console.log("reLaunch success")
      },
      fail: function (res) {
        console.log("reLaunch fail")
      },
      complete: function (res) {
        console.log("reLaunch complete")
      }
    })
  },
```

7）wx.switchTab（OBJECT）：跳转到 tabBar 页面，并关闭其他所有非 tabBar 页面。wx.navigateTo 和 wx.redirectTo 不允许跳转到 tabBar 页面，wx.switchTab 允许跳转到 tabBar 页面。switchTab 的 OBJECT 参数说明如表 4-167 所示。

表 4-167　　switchTab 的 OBJECT 参数说明

参数	类型	必填	说明
url	String	是	需要跳转的 tabBar 页面的路径（需在 app.json 的 tabBar 字段定义的页面），路径后不能带参数
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
//index.js
// 跳转到 tabBar 页面，并关闭其他所有非 tabBar 页面
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
  navButtonClick7: function () {

    wx.navigateTo({
      url: '../test2/test2?id=1',
      success: function (res) {
        console.log("navigateTo success")
      }
    })
  },


// pages/test2/test2.js
Page({
  data: {},
  // 跳转到 tabBar 页面，并关闭其他所有非 tabBar 页面
  tabBarButtonClick: function () {
    wx.switchTab({
      url: '../index/index?id=1',
      success: function (res) {
        console.log("switchTab success")
      },
      fail: function (res) {
        console.log("switchTab fail")
      },
      complete: function (res) {
        console.log("switchTab complete")
      }
    })
  },
})
```

8）wx.navigateBack（OBJECT）：关闭当前页面，返回上一页面或多级页面。使用 wx.navigateTo 方法跳转时，调用该方法的页面会被加入堆栈，而使用 wx.redirectTo 方法则不会。用户可通过 getCurrentPages()获取当前的页面栈并决定需要返回几层。navigateBack 的 OBJECT 参数说明如表 4-168 所示。

表 4-168　　navigateBack 的 OBJECT 参数说明

参数	类型	默认值	说明
delta	Number	1	返回的页面数，如果 delta 大于现有页面数，则返回到首页

使用代码如下所示：

```
//index.js
// 关闭当前页面，返回上一页面或多级页面
  navButtonClick8: function () {
    wx.navigateTo({
      url: '../A/A',
      success: function (res) {
        console.log("navigateTo success")
      }
    })
  },
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
// pages/A/A.js
Page({
  data:{},
   navButtonClick: function () {
    wx.navigateTo({
      url: '../B/B',
      success: function (res) {
        console.log("navigateTo success")
      }
    })
  }
})

// pages/B/B.js
Page({
  data:{},
   navButtonClick: function () {
    wx.navigateTo({
      url: '../C/C',
      success: function (res) {
        console.log("navigateTo success")
      }
    })
  }
})

// pages/C/C.js
Page({
  data: {},
  navButtonClick: function () {
    wx.navigateBack({
      delta: 2 //navigateBack 2 时，将返回 A 页面
    })
  }
})
```

9）wx.setNavigationBarColor(OBJECT)：从基础库 1.4.0 开始支持，设置导航栏背景色。setNavigationBarColor 的 OBJECT 参数说明如表 4-169 所示。

表 4-169　　setNavigationBarColor 的 OBJECT 参数说明

参数名	类型	必填	说明
frontColor	String	是	前景颜色值，包括按钮、标题、状态栏的颜色，仅支持#ffffff 和#000000
backgroundColor	String	是	背景颜色值，有效值为十六进制颜色
animation	Object	否	动画效果
animation.duration	Number	否	动画变化时间，默认 0，单位为 ms
animation.timingFunc	String	否	动画变化方式，默认 linear
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

animation.timingFunc 有效值如表 4-170 所示。

表 4-170 animation.timingFunc 有效值

值	说明
linear	动画从头到尾的速度是相同的
easeIn	动画以低速开始
easeOut	动画以低速结束
easeInOut	动画以低速开始和结束

success 返回参数说明如表 4-171 所示。

表 4-171 success 返回参数说明

参数	类型	说明
ErrMsg	String	调用结果

具体使用代码如下所示：

```
wx.setNavigationBarColor({
frontColor: '#ffffff',
backgroundColor: '#ff0000',
animation: {
  duration: 400,
  timingFunc: 'easeIn'
 }
})
```

4.7.3 设置 tabBar

设置 tabBar 一共有 9 个 API，都是从基础库 1.9.0 开始支持，低版本需做兼容处理。

1）wx.setTabBarBadge(OBJECT)：为 tabBar 某一项的右上角添加文本。setTabBarBadge 的 OBJECT 参数说明如表 4-172 所示。

表 4-172 setTabBarBadge 的 OBJECT 参数说明

参数	类型	必填	说明
index	Number	是	tabBar 的哪一项，从左边算起
text	String	是	显示的文本，超过 3 个字符则显示成“…”
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
wx.setTabBarBadge({
    index: 0,
    text: '1'
})
```

2）wx.removeTabBarBadge(OBJECT)：移除 tabBar 某一项右上角的文本。removeTabBarBadge 的 OBJECT 参数说明，如表 4-173 所示。

表 4-173　　removeTabBarBadge 的 OBJECT 参数说明

参数	类型	必填	说明
index	Number	是	tabBar 的哪一项，从左边算起
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
wx.removeTabBarBadge ({
    index: 0,
})
```

3）wx.showTabBarRedDot(OBJECT)：显示 tabBar 某一项右上角的红点。showTabBarRedDot 的 OBJECT 参数说明，如表 4-174 所示。

表 4-174　　showTabBarRedDot 的 OBJECT 参数说明

参数	类型	必填	说明
index	Number	是	tabBar 的哪一项，从左边算起
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
wx.showTabBarRedDot ({
    index: 0,
})
```

4）wx.hideTabBarRedDot(OBJECT)：隐藏 tabBar 某一项右上角的红点。hideTabBarRedDot 的 OBJECT 参数说明，如表 4-175 所示。

表 4-175　　hideTabBarRedDot 的 OBJECT 参数说明

参数	类型	必填	说明
index	Number	是	tabBar 的哪一项，从左边算起

非卖品，仅供非商业用途或交流学习使用

续表

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
wx.hideTabBarRedDot ({
    index: 0,
})
```

5）onTabItemTap：单击 tab 时触发。使用代码如下所示：

```
Page({
    onTabItemTap(item) {
        console.log(item.index)
        console.log(item.pagePath)
        console.log(item.text)
    }
})
```

6）wx.setTabBarStyle(OBJECT)：动态设置 tabBar 的整体样式。setTabBarStyle 的 OBJECT 参数说明如表 4-176 所示。

表 4-176　　setTabBarStyle 的 OBJECT 参数说明

参数	类型	说明
Color	HexColor	tab 上的文字默认颜色
selectedColor	HexColor	tab 上的文字选中时的颜色
backgroundColor	HexColor	tab 的背景色
borderStyle	String	tabBar 上边框的颜色，仅支持 black/white
success	Function	接口调用成功的回调函数
fail	Function	接口调用失败的回调函数
complete	Function	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
wx.setTabBarStyle({
    color: '#FF0000',
    selectedColor: '#00FF00',
    backgroundColor: '#0000FF',
    borderStyle: 'white'
})
```

7）wx.setTabBarItem(OBJECT)：动态设置 tabBar 某一项的内容。setTabBarItem 的 OBJECT 参数说明如表 4-177 所示。

非卖品，仅供非商业用途或交流学习使用

表 4-177 setTabBarItem 的 OBJECT 参数说明

参数	类型	必填	说明
index	Number	是	tabBar 的哪一项，从左边算起
text	String	否	tab 上的按钮文字
iconPath	String	否	图片路径，icon 大小限制为 40KB，建议尺寸为 81px×81px，当 postion 为 top 时，此参数无效，不支持网络图片
selectedIconPath	String	否	选中时的图片路径，icon 大小限制为 40KB，建议尺寸为 81px×81px，当 postion 为 top 时，此参数无效
success	Function	否	接口调用成功的回调函数

使用代码如下所示：

```
wx.setTabBarItem({
    index: 0,
    text: 'text',
    iconPath: '/path/to/iconPath',
    selectedIconPath: '/path/to/selectedIconPath'
})
```

8）wx.showTabBar(OBJECT)：显示 tabBar。showTabBar 的 OBJECT 参数说明如表 4-178 所示。

表 4-178 showTabBar 的 OBJECT 参数说明

参数	类型	必填	说明
animation	Boolean	否	是否需要动画效果，默认否
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

9）wx.hideTabBar(OBJECT)：隐藏 tabBar。hideTabBar 的 OBJECT 参数说明如表 4-179 所示。

表 4-179 hideTabBar 的 OBJECT 参数说明

参数	类型	必填	说明
animation	Boolean	否	是否需要动画效果，默认否
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

4.7.4 设置置顶信息

1）wx.setTopBarText(OBJECT)：动态设置置顶栏文字内容，只有当前小程序被置顶

非卖品，仅供非商业用途或交流学习使用

时生效。如果当前小程序没有被置顶，也能调用成功，但是不会立即生效，只有在用户将这个小程序置顶后才会换上设置的文字内容。注意：调用成功后，需间隔 5s 才能再次调用此接口，如果在 5s 内再次调用此接口，会回调 fail，errMsg："setTopBarText: fail invoke too frequently"。

setTopBarText 的 OBJECT 参数说明，如表 4-180 所示。

表 4-180　　setTopBarText 的 OBJECT 参数说明

参数	类型	必填	说明
text	String	是	置顶栏文字内容
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
wx.setTopBarText({
    text: 'hello, world!'
})
```

2）wx.setNavigationBarTitle(OBJECT)：动态设置当前页面的标题。setNavigationBarTitle 的 OBJECT 参数说明，如表 4-181 所示。

表 4-181　　setNavigationBarTitle 的 OBJECT 参数说明

参数	类型	必填	说明
title	String	是	页面标题
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
wx.setNavigationBarTitle({
    title: '当前页面'
})
```

3）wx.showNavigationBarLoading()：在当前页面显示导航条加载动画。

4）wx.hideNavigationBarLoading()：隐藏导航条加载动画。

4.7.5　动画

动画 API 实现在页面上展示动画的功能。

非卖品，仅供非商业用途或交流学习使用

wx.createAnimation（OBJECT）：创建一个动画实例 animation，通过调用实例的方法来描述动画，最后通过动画实例的 export 方法导出动画数据传递给组件的 animation 属性。export 方法每次调用后会清除掉之前的动画操作。createAnimation 的 OBJECT 参数说明如表 4-182 所示。

表 4-182　　createAnimation 的 OBJECT 参数说明

参数	类型	必填	说明
duration	Integer	否	动画持续时间，单位为 ms，默认值 400
timingFunction	String	否	定义动画的效果，默认值 linear，有效值：linear、ease、ease-in、ease-in-out、ease-out、step-start、step-end
delay	Integer	否	动画延迟时间，单位为 ms，默认值 0
transformOrigin	String	否	设置 transform-origin，默认为"50% 50% 0"

创建动画实例，代码如下所示：

```
    <!--index.wxml-->
    <view class="section2">4.7.5  动画 </view>
    <view animation="{{animationData}}" style="background:red;height:100rpx;
width:100rpx"> </view>
    <button bindtap="animationButtonClick" class="section">创建动画实例
    </button>


    //index.js
    Page({
      data: {
        animationData: {}
      },
    // 创建动画实例
      animationButtonClick: function () {

        //创建动画实例
        var animation = wx.createAnimation({
          transformOrigin: "50% 50%",
          duration: 1000, //动画持续时间
          timingFunction: 'ease', //动画效果
          delay: 0 //动画延迟时间
        })

        //赋值给 this.animation 并绑定
        this.animation = animation
      },
```

<view/>组件通过 animation="{{animationData}}"来动态绑定动画，wx.createAnimation 来创建动画实例。

animation：动画实例可以调用以下方法来描述动画，调用结束后会返回自身，可以调用多个方法。支持链式调用的写法。animation 的样式如表 4-183 所示。

表 4-183 animation 的样式

方法	参数	说明
opacity	value	透明度，参数范围 0～1
backgroundColor	color	颜色值
width	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值
height	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值
top	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值
left	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值
bottom	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值
right	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值

animation 的旋转如表 4-184 所示。

表 4-184 animation 的旋转

方法	参数	说明
rotate	deg	deg 的范围-180～180，从原点顺时针旋转一个 deg 角度
rotateX	deg	deg 的范围-180～180，在 X 轴旋转一个 deg 角度
rotateY	deg	deg 的范围-180～180，在 Y 轴旋转一个 deg 角度
rotateZ	deg	deg 的范围-180～180，在 Z 轴旋转一个 deg 角度
rotate3d	(x,y,z,deg)	同 transform-function rotate3d

animation 的缩放如表 4-185 所示。

表 4-185 animation 的缩放

方法	参数	说明
scale	sx,[sy]	只有一个参数时，表示在 X 轴、Y 轴同时缩放 sx 倍；有两个参数时，表示在 X 轴缩放 sx 倍，在 Y 轴缩放 sy 倍
scaleX	sx	在 X 轴缩放 sx 倍
scaleY	sy	在 Y 轴缩放 sy 倍
scaleZ	sz	在 Z 轴缩放 sz 倍
scale3d	(sx,sy,sz)	在 X 轴缩放 sx 倍，在 Y 轴缩放 sy 倍，在 Z 轴缩放 sz 倍

animation 的偏移如表 4-186 所示。

表 4-186 animation 的偏移

方法	参数	说明
translate	tx,[ty]	只有一个参数时，表示在 X 轴偏移 tx，单位为 px；有两个参数时，表示在 X 轴偏移 tx，在 Y 轴偏移 ty，单位为 px
translateX	tx	在 X 轴偏移 tx，单位为 px
translateY	ty	在 Y 轴偏移 ty，单位为 px
translateZ	tz	在 Z 轴偏移 tz，单位为 px
translate3d	(tx,ty,tz)	在 X 轴偏移 tx，在 Y 轴偏移 ty，在 Z 轴偏移 tz，单位为 px

animation 的倾斜如表 4-187 所示。

表 4-187 animation 的倾斜

方法	参数	说明
skew	ax,[ay]	参数范围-180～180；只有一个参数时，Y 轴坐标不变，X 轴坐标沿顺时针倾斜 ax 度；有两个参数时，分别在 X 轴倾斜 ax 度，在 Y 轴倾斜 ay 度
skewX	ax	参数范围-180～180；Y 轴坐标不变，X 轴坐标沿顺时针倾斜 ax 度
skewY	ay	参数范围-180～180；X 轴坐标不变，Y 轴坐标沿顺时针倾斜 ay 度

animation 的矩阵变形如表 4-188 所示。

表 4-188 animation 的矩阵变形

方法	参数	说明
matrix	(a,b,c,d,tx,ty)	同 transform-function matrix
matrix3d		同 transform-function matrix3d

调用动画操作方法后要调用 step 来表示一组动画完成，可以在一组动画中调用任意多个动画方法，一组动画中的所有动画会同时开始，一组动画完成后才会进行下一组动画。step 可以传入一个跟 wx.createAnimation 一样的配置参数用于指定当前组动画的配置。

任何动画在使用之前，必须先创建动画实例，然后设置动画描述，通过赋值给实例进行动画操作。具体使用代码如下所示，可以参考注释了解各种情况的使用。

```
// 旋转同时放大
  rotateAndScale: function () {

    //X 轴、Y 轴同时缩放 2 倍，从原点顺时针旋转 45°
    animation.scale(2, 2).rotate(45).step()

    //赋值给 view 绑定的动画
    this.setData({
      animationData: animation.export()
    })
```

```
  },

  // 先旋转后放大
  rotateThenScale: function () {

    this.animation.rotate(45).step()
    this.animation.scale(2, 2).step()
    this.setData({
      animationData: this.animation.export()
    })
  },

  // 旋转的同时放大，然后平移
  rotateAndScaleThenTranslate: function () {

    this.animation.rotate(45).scale(2, 2).step()
    this.animation.translate(100, 100).step({ duration: 1000 })
    this.setData({
      animationData: this.animation.export()
    })
  }
```

4.7.6 下拉刷新

在 config 的 window 选项中开启 enablePullDownRefresh 可以实现下拉刷新。

onPullDownRefresh：在 Page 中定义 onPullDownRefresh 处理函数，可监听该页面用户下拉刷新事件。

```
Page({
onPullDownRefresh: function(){

//处理需要一些时间
……
……
//停止刷新
wx.stopPullDownRefresh()
}
})
```

调用 wx.startPullDownRefresh(OBJECT) API 后触发下拉刷新动画，效果与用户手动下拉刷新一致。startPullDownRefresh 的 OBJECT 参数说明如表 4-189 所示。

表 4-189　　startPullDownRefresh 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-190 所示。

表 4-190　　success 返回参数说明

参数	类型	说明
errMsg	String	接口调用结果

使用代码如下所示：

```
wx.startPullDownRefresh()
```

当处理完数据刷新后，使用 wx.stopPullDownRefresh 可以停止当前页面的下拉刷新。

```
//停止当前页面下拉刷新
wx.stopPullDownRefresh()
```

4.7.7 位置

1）wx.pageScrollTo(OBJECT)：将页面滚动到目标位置，从基础库 1.4.0 开始支持。pageScrollTo 的 OBJECT 参数说明如表 4-191 所示。

表 4-191　　pageScrollTo 的 OBJECT 参数说明

参数	类型	必填	说明
scrollTop	Number	是	滚动到页面的目标位置（单位为 px）

使用代码如下所示：

```
wx.pageScrollTo({
  scrollTop: 0
})
```

2）wx.createSelectorQuery()：WXML 节点信息 API。从基础库 1.4.0 开始支持，返回一个 selectorQuery 对象实例。在这个实例上可以使用 select 等方法选择节点，并使用 boundingClientRect 等方法选择需要查询的信息。

示例代码如下所示：

```
Page({
    queryMultipleNodes: function () {
      var query = wx.createSelectorQuery()
      query.select('#the-id').boundingClientRect()
      query.selectViewport().scrollOffset()
      query.exec(function (res) {
        res[0].top       // #the-id 节点的上边界坐标
        res[1].scrollTop // 显示区域的竖直滚动位置
      })
    }
  })
```

selectorQuery 对象的方法列表如表 4-192 所示。

表 4-192　　selectorQuery 对象的方法列表

方法	参数	说明
select	selector	参考下面详细介绍
selectAll	selector	参考下面详细介绍
selectViewport		参考下面详细介绍
exec	[callback]	参考下面详细介绍

➢ selectorQuery.select(selector)

在当前页面下选择第一个匹配选择器 selector 的节点，返回一个 nodesRef 对象实例，可以用于获取节点信息。

selector 类似于 CSS 的选择器，但仅支持下列语法。

- ID 选择器：#the-id。
- class 选择器（可以连续指定多个）：.a-class.another-class。
- 子元素选择器：.the-parent > #the-child.a-class。
- 多选择器的并集：#a-node, .some-other-nodes。

➢ selectorQuery.selectAll(selector)

在当前页面下选择匹配选择器 selector 的节点，返回一个 nodesRef 对象实例。与 selectorQuery.select(selector)不同的是，它选择所有匹配选择器的节点。

➢ selectorQuery.selectViewport()

选择显示区域，可用于获取显示区域的尺寸、滚动位置等信息，返回一个 nodesRef 对象实例。

➢ nodesRef.boundingClientRect([callback])

添加节点布局位置的查询请求，相对于显示区域，以像素为单位。其功能类似于 DOM 的 getBoundingClientRect。返回值是 nodesRef 对应的 selectorQuery。

返回的节点信息中，每个节点的位置用 left、right、top、bottom、width、height 字段描述。如果提供了 callback 回调函数，在执行 selectQuery 的 exec 方法后，节点信息会在 callback 中返回。

示例代码如下所示：

```
Page({
      getRect: function () {
        wx.createSelectorQuery().select('#the-id').boundingClientRect(function
(rect) {
          rect.id      // 节点的 ID
          rect.dataset // 节点的 dataset
          rect.left    // 节点的左边界坐标
```

```
            rect.right   // 节点的右边界坐标
            rect.top     // 节点的上边界坐标
            rect.bottom  // 节点的下边界坐标
            rect.width   // 节点的宽度
            rect.height  // 节点的高度
          }).exec()
        },
        getAllRects: function () {
          wx.createSelectorQuery().selectAll('.a-class').boundingClientRect
(function (rects) {
            rects.forEach(function (rect) {
              rect.id      // 节点的 ID
              rect.dataset // 节点的 dataset
              rect.left    // 节点的左边界坐标
              rect.right   // 节点的右边界坐标
              rect.top     // 节点的上边界坐标
              rect.bottom  // 节点的下边界坐标
              rect.width   // 节点的宽度
              rect.height  // 节点的高度
            })
          }).exec()
        }
      })
```

➢ nodesRef.scrollOffset([callback])

添加节点滚动位置的查询请求，以像素为单位。节点必须是 scroll-view 或者 viewport。返回值是 nodesRef 对应的 selectorQuery。

返回的节点信息中，每个节点的滚动位置用 scrollLeft、scrollHeight 字段描述。如果提供了 callback 回调函数，在执行 selectQuery 的 exec 方法后，节点信息将会在 callback 中返回。

示例代码如下所示：

```
Page({
      getScrollOffset: function () {
        wx.createSelectorQuery().selectViewport().scrollOffset(function (res) {
          res.id      // 节点的 ID
          res.dataset // 节点的 dataset
          res.scrollLeft // 节点的水平滚动位置
          res.scrollTop  // 节点的竖直滚动位置
        }).exec()
      }
    })
```

➢ nodesRef.fields(fields, [callback])

获取节点的相关信息，需要获取的字段在 fields 中指定。返回值是 nodesRef 对应的 selectorQuery。selectorQuery 可指定获取的字段如表 4-193 所示。

表 4-193　　selectorQuery 可指定获取的字段

字段名	默认值	说明
id	否	是否返回节点 id

续表

字段名	默认值	说明
dataset	否	是否返回节点 dataset
rect	否	是否返回节点布局位置（left、right、top、bottom）
size	否	是否返回节点尺寸（width、height）
scrollOffset	否	是否返回节点的 scrollLeft、scrollTop，节点必须是 scroll-view 或者 viewport
properties	[]	指定属性名列表，返回节点对应属性名的当前属性值（只能获得组件文档中标注的常规属性值，id、class、style 和事件绑定的属性值不可获取）

示例代码如下所示：

```
Page({
    getFields: function () {
      wx.createSelectorQuery().select('#the-id').fields({
        dataset: true,
        size: true,
        scrollOffset: true,
        properties: ['scrollX', 'scrollY']
      }, function (res) {
        res.dataset    // 节点的 dataset
        res.width      // 节点的宽度
        res.height     // 节点的高度
        res.scrollLeft // 节点的水平滚动位置
        res.scrollTop  // 节点的竖直滚动位置
        res.scrollX    // 节点 scroll-x 属性的当前值
        res.scrollY    // 节点 scroll-x 属性的当前值
      }).exec()
    }
  })
```

- selectorQuery.exec([callback])

执行所有的请求，请求结果按请求次序构成数组，在 callback 的第一个参数中返回。

4.8 绘图

绘图 API 可以实现在画布上绘制各种线条和图形，并保存图片。使用步骤分为 4 步。

第 1 步：在.wxml 文件中，创建画布组件<canvas/>。画布的位置可以通过<view/>组件来控制。具体代码如下所示：

```
<!--index.wxml-->

<!--创建一个容器视图，宽 100%，高 300-->
<view class="flex-wrp" style="height: 300px;flex-direction:row;justify-content:
center;align-items: center">

<!-在容器 view 内添加一个 class="flex-item" 宽 80%，高 300-->
<view class="flex-item">
```

```
<!--添加画布，大小和容器内子视图 view 一样-->
<canvas canvas-id="myCanvas" style="height: 100%;width:100%;border: 1px solid;"/>

</view>
</view>

/**index.wxss**/
.flex-wrp{
    display:flex;
    background-color: #FFFFFF;
}
.flex-item{
    width: 90%;
    height: 300px;
    background-color: #ffffff;
}
```

第 2 步：在.js 文件中，创建一个 canvas 绘图上下文 canvasContext。代码如下所示：

```
// 通过<canvas/>组件 id'myCanvas'来创建的一个上下文对象 ctx
    const ctx = wx.createCanvasContext('myCanvas')
```

第 3 步：使用 canvas 绘图上下文进行绘图描述。代码如下所示：

```
// 设置填充色为红色
    ctx.setFillStyle('red')

// 用 fillRect(x, y, width, height) 方法画一个矩形
    ctx.fillRect(10, 10, 150, 75)
```

第 4 步：通知<canvas/> 组件，进行绘图。代码如下所示：

```
// 通知 <canvas/> 组件，进行绘图
    ctx.draw()
```

上面是以绘制一个红色矩形为例，实现效果如图 4-4 所示。

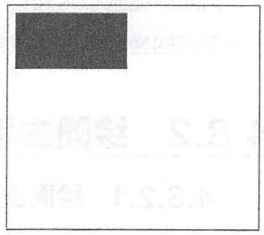


图 4-4 绘制红色矩形

4.8.1 坐标系介绍（coordinates）

canvas 坐标系是一个二维的网格。左上角的坐标为（0, 0）。ctx.fillRect（10, 10, 150, 75）绘制矩形的意思就是从左上角（10, 10）开始，画一个 150px × 75px 的矩形。可以在<canvas/>中加上触摸事件，来观测它的坐标系。代码如下所示：

```
<!--index.wxml-->
<!--添加画布 -->
<canvas canvas-id="myCanvas"
style="height: 100%;width:100%;border: 1px solid;"
bindtouchstart="start"
bindtouchmove="move"
bindtouchend="end"/>
<view class="section" hidden="{{hidden}}">
  Coordinates: ({{x}}, {{y}})
```

```
</view>

//index.js
Page({
  data: {
    x: 0,
    y: 0,
    hidden: true
  },

  //触摸屏幕
  start: function (e) {
    this.setData({
      hidden: false,
      x: e.touches[0].x,
      y: e.touches[0].y
    })
  },

  //在画布内移动
  move: function (e) {
    this.setData({
      x: e.touches[0].x,
      y: e.touches[0].y
    })
  },

  //释放，离开画布
  end: function (e) {
    this.setData({
      hidden: true
    })
  }
})
```

当用户在画布内触摸屏幕时开始计时，在移动的过程中显示触摸点的坐标。显示效果如图 4-5 所示。

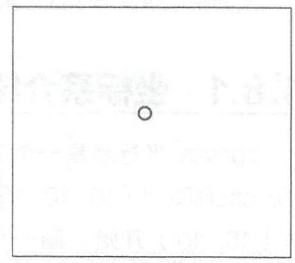


图 4-5 显示触摸点坐标

4.8.2 绘图主接口

4.8.2.1 绘图上下文

wx.createCanvasContext (canvasId): 创建 canvas 绘图上下文，指定 canvasId，该绘图上下文只作用于对应的 <canvas/>组件内。createCanvasContext 的具体参数说明如表 4-194 所示。

表 4-194 createCanvasContext 的参数说明

参数	类型	说明
canvasId	String	画布标识，传入定义在 <canvas/> 的 canvas-id

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

使用代码如下所示：

```
// 通过<canvas/>组件 id'myCanvas'来创建的一个上下文对象 ctx
   const ctx = wx.createCanvasContext('myCanvas')
```

4.8.2.2 创建并返回绘图上下文

wx.createContext 创建并返回绘图上下文。微信不推荐使用该 API，在此不做介绍。

4.8.2.3 绘制画布

drawCanvas：用所提供的 actions 在所给的 canvas-id 对应的 canvas 上进行绘图。微信不推荐使用该 API。drawCanvas 的具体参数说明如表 4-195 所示。

表 4-195　　drawCanvas 的参数说明

参数	类型	说明
canvasId	String	画布标识，传入 <canvas/> 的 canvas-id
actions	Array	绘图动作数组，由 wx.createContext 创建的 context，调用 getActions 方法导出绘图动作数组
reserve	Boolean	（可选）本次绘制是否接着上一次绘制，如果 reserve 参数为 false，则在本次调用 drawCanvas 绘制之前 native 层应先清空画布再继续绘制；如果 reserver 参数为 true，则保留当前画布上的内容，本次调用 drawCanvas 绘制的内容覆盖在上面。默认 false

使用代码如下所示：

```
// 4.8.2.3 绘制画布 drawCanvas
  textButtonClick1: function () {

    //使用 wx.createContext 获取绘图上下文 context
    var context = wx.createContext();
    //颜色
    context.setStrokeStyle("#ff5400");
    //线宽度
    context.setLineWidth(6);
    //矩形
    context.rect(10, 10, 200, 200);
    //开始画线
    context.stroke()

    //绘制圆
    context.setStrokeStyle("#ff0000");
    context.setLineWidth(2)
    context.moveTo(160, 100)
    //x=100,y=100,开始=0°，结束=360°，逆时针
    context.arc(100, 100, 60, 0, 2 * Math.PI, true);

    //绘制嘴巴
    context.moveTo(140, 100);
    context.arc(100, 100, 40, 0, Math.PI, false);

    //绘制左眼
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    context.moveTo(85, 80);
    context.arc(80, 80, 5, 0, 2 * Math.PI, true);

    // 绘制右眼
    context.moveTo(125, 80);
    context.arc(120, 80, 5, 0, 2 * Math.PI, true);
    context.stroke();

    //调用 wx.drawCanvas
    wx.drawCanvas({

      //通过 canvasId 指定画布
      canvasId: 'myCanvas',

      //通过 actions 指定绘制行为,获取绘图动作数组
      actions: context.getActions()
    });
  },
```

4.8.2.4 导出图片

canvasToTempFilePath（OBJECT）：把当前画布的内容导出生成图片，并返回文件路径。canvasToTempFilePath 的 OBJECT 参数说明如表 4-196 所示。

表 4-196 canvasToTempFilePath 的 OBJECT 参数说明

参数	类型	必填	说明
x	Number	否	画布 X 轴起点（默认 0）
y	Number	否	画布 Y 轴起点（默认 0）
width	Number	否	画布宽度（默认为 canvas 宽度-x）
height	Number	否	画布高度（默认为 canvas 高度-y）
destWidth	Number	否	输出图片宽度（默认为 width）
destHeight	Number	否	输出图片高度（默认为 height）
canvasId	String	是	画布标识，传入 <canvas/> 的 canvas-id
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
// 4.8.2.4 导出图片
  textButtonClick2: function () {
    wx.canvasToTempFilePath({
      x: 100,
      y: 200,
      width: 50,
      height: 50,
      destWidth: 100,
      destHeight: 100,
      canvasId: 'myCanvas',
      success: function (res) {
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        console.log(res.tempFilePath)
      },
      fail: function (res) {
        console.log("canvasToTempFilePath fail")
      },
      complete: function (res) {
        console.log("canvasToTempFilePath complete")
      },
    })
  },
```

4.8.3 填充颜色、线条、阴影

4.8.3.1 设置填充样式

setFillStyle 用来设置填充颜色。如果没有设置 setFillStyle，默认颜色为黑色。setFillStyle 的参数如表 4-197 所示。

表 4-197 setFillStyle 的参数

参数	类型	定义
color	Color	Gradient Object

代码如下所示，显示效果如图 4-6 所示。

```
// 4.8.3.1 设置填充样式
    const ctx = wx.createCanvasContext('testCanvas')
    ctx.setFillStyle('#ffa589')
    ctx.fillRect(10, 10, 150, 75) //绘制矩形
    ctx.draw()
  },
```

4.8.3.2 设置线条样式

setStrokeStyle 用来设置边框颜色。如果没有设置 setStrokeStyle，默认颜色为黑色。setStrokeStyle 的参数如表 4-198 所示。

表 4-198 setStrokeStyle 的参数

参数	类型	定义
color	Color	Gradient Object

使用代码如下所示，显示效果如图 4-7 所示。

```
// 4.8.3.2 设置线条样式
    const ctx = wx.createCanvasContext('testCanvas')
    ctx.setStrokeStyle('red')
    ctx.strokeRect(10, 10, 150, 75) //绘制矩形
    ctx.draw()
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

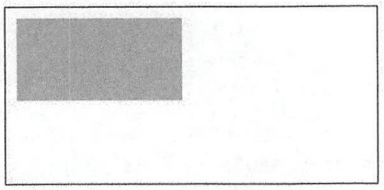


图 4-6 设置填充颜色

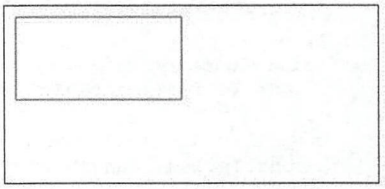


图 4-7 设置线条样式

4.8.3.3 设置阴影样式

setShadow 用来设置阴影样式。如果没有设置，offsetX 默认值为 0，offsetY 默认值为 0，blur 默认值为 0，color 默认值为黑色。setShadow 的参数如表 4-199 所示。

表 4-199 setShadow 的参数

参数	类型	范围	定义
offsetX	Number		阴影相对于形状在水平方向的偏移
offsetY	Number		阴影相对于形状在竖直方向的偏移
blur	Number	0～100	阴影的模糊级别，数值越大越模糊
color	Color		阴影的颜色

使用代码如下所示，显示效果如图 4-8 所示。

```
// 4.8.3.3 设置阴影样式
    const ctx = wx.createCanvasContext('testCanvas')
    ctx.setFillStyle('#ffa589')
    ctx.setShadow(10, 50, 50, '#0098b6') //阴影区域和颜色
    ctx.fillRect(10, 10, 150, 75)  //绘制矩形
    ctx.draw()
```

4.8.4 渐变

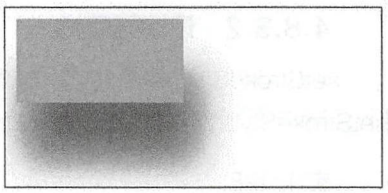


图 4-8 设置阴影样式

4.8.4.1 创建线性渐变

createLinearGradient 用于创建一个线性的渐变颜色。需要使用 addColorStop()来指定渐变点，至少要两个点，范围 0～1。createLinearGradient 的参数如表 4-200 所示。

表 4-200 createLinearGradient 的参数

参数	类型	定义
x0	Number	起点的 X 轴坐标
y0	Number	起点的 Y 轴坐标
x1	Number	终点的 X 轴坐标
y1	Number	终点的 Y 轴坐标

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

使用代码如下所示，显示效果如图 4-9 所示。

```
// 4.8.4.1 创建线性渐变
    const ctx = wx.createCanvasContext('testCanvas')
    //创建线性渐变区域
    const grd = ctx.createLinearGradient(0, 0, 200, 0)
//设置渐变点
    //可以设置 0~1 之间任意数，例如：0.2、0.5、0.6
    grd.addColorStop(0, 'red')
    grd.addColorStop(1, 'white')
    //填充颜色
    ctx.setFillStyle(grd)
    ctx.fillRect(10, 10, 150, 80) //绘制矩形
    ctx.draw()
```

4.8.4.2 创建圆形渐变

createCircularGradient 用于创建一个圆形的渐变颜色。起点在圆心，终点在圆环。需要使用 addColorStop()来指定渐变点，至少要两个点。createCircularGradient 的参数如表 4-201 所示。

表 4-201 createCircularGradient 的参数

参数	类型	定义
x	Number	圆心的 *X* 轴坐标
y	Number	圆心的 *Y* 轴坐标
r	Number	圆的半径

使用代码如下所示，显示效果如图 4-10 所示。

```
// 4.8.4.2 创建圆形渐变
    const ctx = wx.createCanvasContext('testCanvas')

    //创建圆形区域
    const grd = ctx.createCircularGradient(75, 50, 50)
    //设置渐变点
    grd.addColorStop(0, '#ff5400')
    grd.addColorStop(1, '#ffffff')

    // 填充颜色
    ctx.setFillStyle(grd)
    ctx.fillRect(10, 10, 150, 200)
    ctx.draw()
```

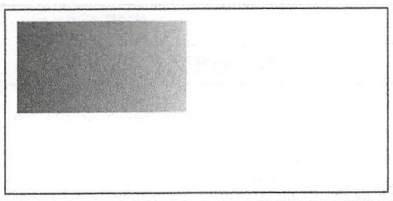


图 4-9 创建线性渐变

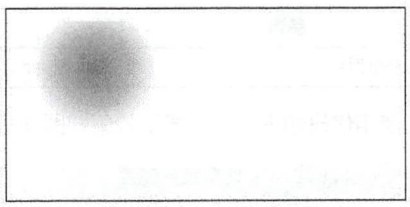


图 4-10 创建圆形渐变

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

4.8.4.3 创建颜色渐变

addColorStop 用于创建一个颜色的渐变点。小于最小 stop 的部分会按最小 stop 的 color 来渲染，大于最大 stop 的部分会按最大 stop 的 color 来渲染。需要使用 addColorStop()来指定渐变点，至少要两个点。addColorStop 的参数如表 4-202 所示。

表 4-202　　addColorStop 的参数

参数	类型	定义
stop	Number(0～1)	表示渐变点在起点和终点之间的位置
color	Color	渐变点的颜色

使用代码如下所示，显示效果如图 4-11 所示。

```
// 4.8.4.3 创建颜色渐变
    const ctx = wx.createCanvasContext('testCanvas')
    //创建线性渐变区域
    const grd = ctx.createLinearGradient(30, 10, 120, 10)
    //设置渐变点
    grd.addColorStop(0, 'red')
    grd.addColorStop(0.16, 'orange')
    grd.addColorStop(0.33, 'yellow')
    grd.addColorStop(0.5, 'green')
    grd.addColorStop(0.66, 'cyan')
    grd.addColorStop(0.83, 'blue')
    grd.addColorStop(1, 'purple')

    //填充颜色
    ctx.setFillStyle(grd)
    //绘制矩形
    ctx.fillRect(10, 10, 150, 80)
    ctx.draw()
```

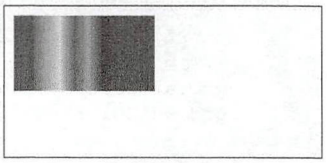


图 4-11　创建颜色渐变

4.8.5 线条样式

4.8.5.1 设置线条宽度

setLineWidth 用于设置线条的宽度。setLineWidth 的参数说明如表 4-203 所示。

表 4-203　　setLineWidth 的参数说明

参数	类型	说明
lineWidth	Number	线条的宽度（单位为 px）

使用代码如下所示，显示效果如图 4-12 所示。

```
// 4.8.5.1 设置线条宽度
    const ctx = wx.createCanvasContext('testCanvas')
    ctx.setStrokeStyle('#ffa589')
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    ctx.beginPath()
    ctx.moveTo(10, 10)
    ctx.lineTo(150, 10)
    ctx.stroke()

    ctx.beginPath()
    ctx.setLineWidth(5)
    ctx.moveTo(10, 30)
    ctx.lineTo(150, 30)
    ctx.stroke()

    ctx.beginPath()
    ctx.setLineWidth(10)
    ctx.moveTo(10, 50)
    ctx.lineTo(150, 50)
    ctx.stroke()

    ctx.beginPath()
    ctx.setLineWidth(15)
    ctx.moveTo(10, 70)
    ctx.lineTo(150, 70)
    ctx.stroke()

    ctx.draw()
```

4.8.5.2 设置线条端点样式

setLineCap 用于设置线条的端点样式。setLineCap 的参数说明如表 4-204 所示。

表 4-204 setLineCap 的参数说明

参数	类型	范围	说明
lineCap	String	butt、round、square	线条的端点样式

使用代码如下所示，显示效果如图 4-13 所示。

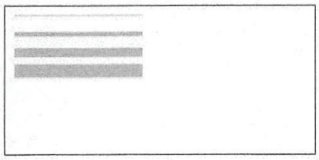


图 4-12 设置线条宽度

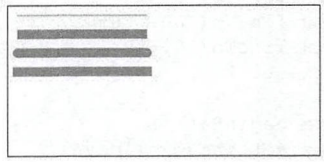


图 4-13 设置线条端点样式

```
// 4.8.5.2 设置线条端点样式
    const ctx = wx.createCanvasContext('testCanvas')
    ctx.setStrokeStyle('#ff5400')
    ctx.beginPath()
    ctx.moveTo(10, 10)
    ctx.lineTo(150, 10)
    ctx.stroke()

    ctx.beginPath()
    ctx.setLineCap('butt')
    ctx.setLineWidth(10)
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    ctx.moveTo(10, 30)
    ctx.lineTo(150, 30)
    ctx.stroke()

    ctx.beginPath()
    ctx.setLineCap('round')
    ctx.setLineWidth(10)
    ctx.moveTo(10, 50)
    ctx.lineTo(150, 50)
    ctx.stroke()

    ctx.beginPath()
    ctx.setLineCap('square')
    ctx.setLineWidth(10)
    ctx.moveTo(10, 70)
    ctx.lineTo(150, 70)
    ctx.stroke()

    ctx.draw()
```

4.8.5.3 设置线条交点样式

setLineJoin 用于设置线条的交点样式。setLineJoin 的参数说明如表 4-205 所示。

表 4-205 setLineJoin 的参数说明

参数	类型	范围	说明
lineJoin	String	bevel、round、miter	线条的交点样式

使用代码如下所示，显示效果如图 4-14 所示。

```
// 4.8.5.3 设置线条交点样式
    const ctx = wx.createCanvasContext('testCanvas')
    ctx.setStrokeStyle('#ff5400')
    ctx.beginPath()
    ctx.moveTo(10, 10)
    ctx.lineTo(100, 50)
    ctx.lineTo(10, 90)
    ctx.stroke()

    ctx.beginPath()
    ctx.setLineJoin('bevel')
    ctx.setLineWidth(10)
    ctx.moveTo(50, 10)
    ctx.lineTo(140, 50)
    ctx.lineTo(50, 90)
    ctx.stroke()

    ctx.beginPath()
    ctx.setLineJoin('round')
    ctx.setLineWidth(10)
    ctx.moveTo(90, 10)
    ctx.lineTo(180, 50)
    ctx.lineTo(90, 90)
    ctx.stroke()
```

非卖品，仅供非商业用途或交流学习使用

```
    ctx.beginPath()
    ctx.setLineJoin('miter')
    ctx.setLineWidth(10)
    ctx.moveTo(130, 10)
    ctx.lineTo(220, 50)
    ctx.lineTo(130, 90)
    ctx.stroke()

    ctx.draw()
```

4.8.5.4 设置最大斜接长度

setMiterLimit 用于设置最大斜接长度，斜接长度指的是在两条线交会处内角和外角之间的距离。当 setLineJoin()为 miter 时才有效。超过最大倾斜长度的，连接处将以 lineJoin 为 bevel 来显示。setMiterLimit 的参数说明如表 4-206 所示。

表 4-206　　setMiterLimit 的参数说明

参数	类型	说明
miterLimit	Number	最大斜接长度

使用代码如下所示，显示效果如图 4-15 所示。

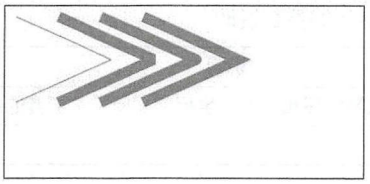


图 4-14　设置线条交点样式

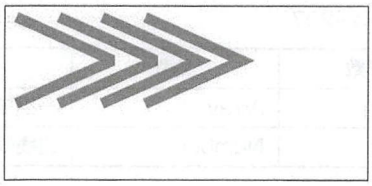


图 4-15　设置最大斜接长度

```
// 4.8.5.4  设置最大斜接长度
    const ctx = wx.createCanvasContext('testCanvas')
    ctx.setStrokeStyle('#ff5400')
    ctx.beginPath()
    ctx.setLineWidth(10)
    ctx.setLineJoin('miter')
    ctx.setMiterLimit(1)
    ctx.moveTo(10, 10)
    ctx.lineTo(100, 50)
    ctx.lineTo(10, 90)
    ctx.stroke()

    ctx.beginPath()
    ctx.setLineWidth(10)
    ctx.setLineJoin('miter')
    ctx.setMiterLimit(2)
    ctx.moveTo(50, 10)
    ctx.lineTo(140, 50)
    ctx.lineTo(50, 90)
    ctx.stroke()
```

非卖品，仅供非商业用途或交流学习使用

```
ctx.beginPath()
ctx.setLineWidth(10)
ctx.setLineJoin('miter')
ctx.setMiterLimit(3)
ctx.moveTo(90, 10)
ctx.lineTo(180, 50)
ctx.lineTo(90, 90)
ctx.stroke()

ctx.beginPath()
ctx.setLineWidth(10)
ctx.setLineJoin('miter')
ctx.setMiterLimit(4)
ctx.moveTo(130, 10)
ctx.lineTo(220, 50)
ctx.lineTo(130, 90)
ctx.stroke()

ctx.draw()
```

4.8.5.5 设置虚线

canvasContext.setLineDash 从基础库 1.6.0 开始支持，可以设置虚线。setLineDash 的参数说明如表 4-207 所示。

表 4-207　　setLineDash 的参数说明

参数	类型	说明
pattern	Array	一组描述交替绘制线段和间距（坐标空间单位）长度的数字
offset	Number	虚线偏移量

使用代码如下所示：

```
const ctx = wx.createCanvasContext('testCanvas')
  ctx.setLineDash([20, 40], 5);
  ctx.beginPath();
  ctx.moveTo(0, 100);
  ctx.lineTo(400, 100);
  ctx.stroke();
  ctx.draw()
```

20 是指一段虚线的长度，40 是指虚线之间的间距。

效果图如图 4-16 所示。

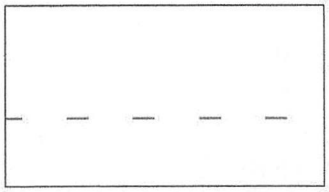


图 4-16　虚线效果

非卖品，仅供非商业用途或交流学习使用

4.8.6 矩形

4.8.6.1 创建矩形

rect 用于创建一个矩形，用 fill 或者 stroke 方法将矩形真正地画到 canvas 中。rect 的参数说明如表 4-208 所示。

表 4-208 rect 的参数说明

参数	类型	说明
x	Number	矩形路径左上角的 X 轴坐标
y	Number	矩形路径左上角的 Y 轴坐标
width	Number	矩形路径的宽度
height	Number	矩形路径的高度

使用代码如下所示，显示效果如图 4-17 所示。

```
// 4.8.6.1 创建矩形
    const ctx = wx.createCanvasContext('testCanvas')
    ctx.rect(10, 10, 150, 75)
    ctx.setFillStyle('#ff5400')
    ctx.fill()
    ctx.draw()
```

4.8.6.2 填充矩形

fillRect 用于填充一个矩形。setFillStyle 用于设置矩形的填充色，如果没有设置，默认是黑色。fillRect 的参数说明如表 4-209 所示。

表 4-209 fillRect 的参数说明

参数	类型	说明
x	Number	矩形路径左上角的 X 轴坐标
y	Number	矩形路径左上角的 Y 轴坐标
width	Number	矩形路径的宽度
height	Number	矩形路径的高度

使用代码如下所示，显示效果如图 4-18 所示。

```
// 4.8.6.2 填充矩形
    const ctx = wx.createCanvasContext('testCanvas')
    ctx.setFillStyle('#ff5400')
    ctx.fillRect(10, 10, 150, 75)
    ctx.draw()
```

4.8.6.3 画一个矩形

strokeRect 用于画一个矩形（非填充）。setFillStroke 用于设置矩形线条的颜色，如果没

有设置，默认是黑色。strokeRect 的参数说明如表 4-210 所示。

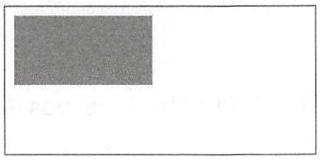


图 4-17 创建矩形

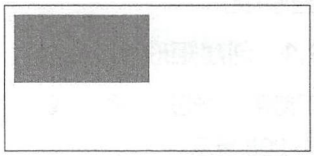


图 4-18 填充矩形

表 4-210 strokeRect 的参数说明

参数	类型	说明
x	Number	矩形路径左上角的 *X* 轴坐标
y	Number	矩形路径左上角的 *Y* 轴坐标
width	Number	矩形路径的宽度
height	Number	矩形路径的高度

使用代码如下所示，显示效果如图 4-19 所示。

```
// 4.8.6.3 画一个矩形
    const ctx = wx.createCanvasContext('testCanvas')
    ctx.setStrokeStyle('#ff5400')
    ctx.strokeRect(10, 10, 150, 75)
    ctx.draw()
```

4.8.6.4 清除该矩形区域内的内容

clearRect 用于清除画布上在该矩形区域内的内容。clearRect 并非画一个白色的矩形在地址区域，而是清空。为了有直观感受，我们对 canvas 加了一层背景色。代码如下所示：

```
<canvas canvas-id="myCanvas" style="border: 1px solid; background: #ffffff;"/>
```

clearRect 的参数说明如表 4-211 所示。

表 4-211 clearRect 的参数说明

参数	类型	说明
x	Number	矩形区域左上角的 *X* 轴坐标
y	Number	矩形区域左上角的 *Y* 轴坐标
width	Number	矩形区域的宽度
height	Number	矩形区域的高度

使用代码如下所示，显示效果如图 4-20 所示。

```
// 4.8.6.4 清除该矩形区域内的内容
    const ctx = wx.createCanvasContext('myCanvas1')
    //先绘制两个矩形
    ctx.setFillStyle('red')
    ctx.fillRect(0, 0, 150, 200)
    ctx.setFillStyle('blue')
    ctx.fillRect(150, 0, 150, 200)
```

```
    //清空
    ctx.clearRect(10, 10, 150, 75)
    ctx.draw()
```

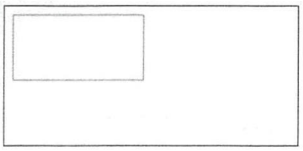


图 4-19 画一个矩形

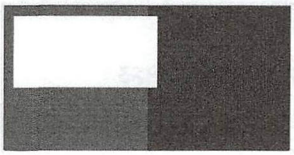


图 4-20 清除画布上该矩形区域内的内容

4.8.7 路径

4.8.7.1 填充路径

fill 用于对当前路径中的内容进行填充。默认的填充色为黑色。如果当前路径没有闭合，fill 方法会将起点和终点进行连接，然后填充。代码如下所示，显示效果如图 4-21 所示。

```
// 4.8.7.1 填充路径
    const ctx = wx.createCanvasContext('testCanvas')
    ctx.moveTo(10, 10)
    ctx.lineTo(100, 10)
    ctx.lineTo(100, 100)
    ctx.fill()
    ctx.draw()
```

fill()填充的路径是从 beginPath()开始计算，但是不会将 fillRect()包含进去，具体代码如下所示，显示效果如图 4-22 所示。

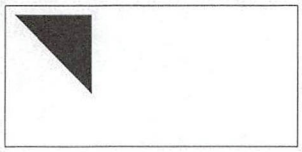


图 4-21 起点和终点连接后填充

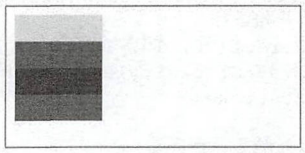


图 4-22 beginPath()开始计算填充

```
const ctx = wx.createCanvasContext('testCanvas')
    // 开始路径
    ctx.rect(10, 10, 100, 30)
    ctx.setFillStyle('yellow')
    ctx.fill()

    // 开始另一个路径
    ctx.beginPath()
    ctx.rect(10, 40, 100, 30)

    // 只填充此矩形，而不填充当前路径
    ctx.setFillStyle('blue')
    ctx.fillRect(10, 70, 100, 30)

    ctx.rect(10, 100, 100, 30)
```

```
    // 填充当前路径
    ctx.setFillStyle('red')
    ctx.fill()
    ctx.draw()
```

4.8.7.2 描边路径

stroke 用于画出当前路径的边框。默认颜色为黑色。如果当前路径没有闭合，fill 方法会将起点和终点进行连接，然后描边。代码如下所示，显示效果如图 4-23 所示。

```
// 4.8.7.2 描边路径
    const ctx = wx.createCanvasContext('testCanvas')
    ctx.moveTo(10, 10)
    ctx.lineTo(100, 10)
    ctx.lineTo(100, 100)
    ctx.stroke()
    ctx.draw()
```

stroke()描绘的路径是从 beginPath()开始计算，但是不会将 strokeRect()包含进去。代码如下所示，显示效果如图 4-24 所示。

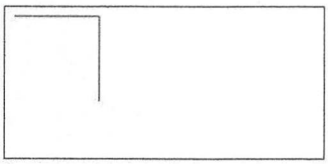


图 4-23 在起点和终点的一侧描边

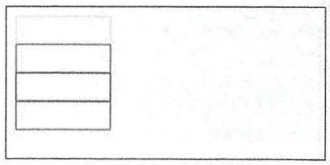


图 4-24 beginPath()开始计算描边

```
const ctx = wx.createCanvasContext('testCanvas')
    // 开始路径
    ctx.rect(10, 10, 100, 30)
    ctx.setStrokeStyle('yellow')
    ctx.stroke()

    // 开始另一个路径
    ctx.beginPath()
    ctx.rect(10, 40, 100, 30)

    // 只填充此矩形，而不填充当前路径
    ctx.setStrokeStyle('blue')
    ctx.strokeRect(10, 70, 100, 30)

    ctx.rect(10, 100, 100, 30)

    // 填充当前路径
    ctx.setStrokeStyle('red')
    ctx.stroke()
    ctx.draw()
```

4.8.7.3 开始路径

beginPath 用于创建一个路径，需要调用 fill 或者 stroke 对路径进行填充或描边。在最开始

的时候相当于调用了一次 beginPath()。同一个路径内多次调用 setFillStyle()、setStrokeStyle()、setLineWidth()等设置，以最后一次调用为准。

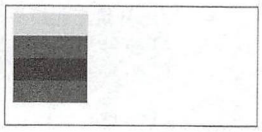


图 4-25 开始路径

使用代码如下所示，显示效果如图 4-25 所示。

```
// 4.8.7.3 开始路径
    const ctx = wx.createCanvasContext('testCanvas')
    // 开始路径
    ctx.rect(10, 10, 100, 30)
    ctx.setFillStyle('yellow')
    ctx.fill()

    // 开始另一个路径
    ctx.beginPath()
    ctx.rect(10, 40, 100, 30)

    // 只填充此矩形，而不填充当前路径
    ctx.setFillStyle('blue')
    ctx.fillRect(10, 70, 100, 30)

    ctx.rect(10, 100, 100, 30)

    // 填充当前路径
    ctx.setFillStyle('red')
    ctx.fill()
    ctx.draw()
```

4.8.7.4 关闭路径

closePath 用于关闭一个路径。关闭路径会连接起点和终点。如果关闭路径后没有调用 fill()或者 stroke()并开启了新的路径，那之前的路径将不会被渲染。示例 1 代码如下所示，显示效果如图 4-26 所示。

```
// 4.8.7.4 关闭路径
    const ctx = wx.createCanvasContext('testCanvas')
    ctx.moveTo(10, 10)
    ctx.lineTo(100, 10)
    ctx.lineTo(100, 100)
    ctx.closePath()
    ctx.stroke()
    ctx.draw()
```

示例 2 代码如下所示，显示效果如图 4-27 所示。

```
const ctx = wx.createCanvasContext('testCanvas')
    // 开始路径
    ctx.rect(10, 10, 100, 30)
    ctx.closePath()

    // 开始另一个路径
    ctx.beginPath()
    ctx.rect(10, 40, 100, 30)
```

```
// 只填充此矩形，而不填充当前路径
ctx.setFillStyle('blue')
ctx.fillRect(10, 70, 100, 30)

ctx.rect(10, 100, 100, 30)

// 填充当前路径
ctx.setFillStyle('red')
ctx.fill()
ctx.draw()
```

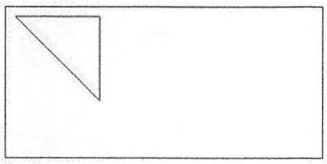


图 4-26 关闭路径，示例 1

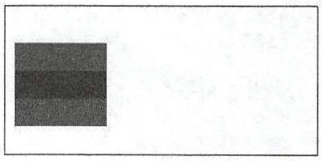


图 4-27 关闭路径，示例 2

4.8.7.5 moveTo

moveTo 用于把路径移动到画布中的指定点，不创建线条。可以使用 stroke 方法来画线条。moveTo 的参数说明如表 4-212 所示。

表 4-212 moveTo 的参数说明

参数	类型	说明
x	Number	目标位置的 X 轴坐标
y	Number	目标位置的 Y 轴坐标

使用代码如下所示，显示效果如图 4-28 所示。

```
// 4.8.7.5 moveTo
    const ctx = wx.createCanvasContext('testCanvas')
    ctx.moveTo(10, 10)
    ctx.lineTo(100, 10)

    ctx.moveTo(10, 50)
    ctx.lineTo(100, 50)
    ctx.stroke()
    ctx.draw()
```

4.8.7.6 lineTo

lineTo 用于增加一个新点，然后创建一条从上次指定点到目标点的线。可以使用 stroke 方法来画线条。lineTo 的参数如表 4-213 所示。

表 4-213 lineTo 的参数

参数	类型	说明
x	Number	目标位置的 X 轴坐标
y	Number	目标位置的 Y 轴坐标

使用代码如下所示，显示效果如图 4-29 所示。

```
// 4.8.7.6 lineTo
    const ctx = wx.createCanvasContext('testCanvas')
    ctx.moveTo(10, 10)
    ctx.rect(10, 10, 100, 50)
    ctx.lineTo(110, 60)
    ctx.stroke()
    ctx.draw()
```

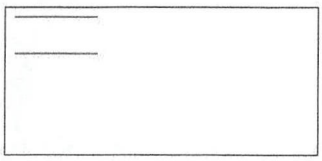


图 4-28 moveTo

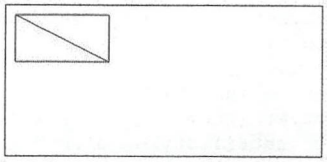


图 4-29 lineTo

4.8.7.7 画弧线

arc 用于画一条弧线。创建一个圆可以用 arc 方法指定起始弧度为 0，终止弧度为 2 * Math.PI。可以使用 stroke 或者 fill 方法在 canvas 中画弧线。arc 的参数如表 4-214 所示。

表 4-214 arc 的参数

参数	类型	说明
x	Number	圆的 *X* 轴坐标
y	Number	圆的 *Y* 轴坐标
r	Number	圆的半径
sAngle	Number	起始弧度（*X* 轴），单位为弧度
eAngle	Number	终止弧度
counterclockwise	Boolean	可选，指定弧度的方向是逆时针还是顺时针。默认是 false，即顺时针

使用代码如下所示，显示效果如图 4-30 所示。

```
// 4.8.7.7 画弧线
    const ctx = wx.createCanvasContext('testCanvas')

    // 画坐标
    ctx.arc(100, 75, 50, 0, 2 * Math.PI)
    ctx.setFillStyle('#EEEEEE')
    ctx.fill()

    ctx.beginPath()
    ctx.moveTo(40, 75)
    ctx.lineTo(160, 75)
    ctx.moveTo(100, 15)
    ctx.lineTo(100, 135)
    ctx.setStrokeStyle('#AAAAAA')
    ctx.stroke()
```

```
ctx.setFontSize(12)
ctx.setFillStyle('black')
ctx.fillText('0', 165, 78)
ctx.fillText('0.5*PI', 83, 145)
ctx.fillText('1*PI', 15, 78)
ctx.fillText('1.5*PI', 83, 10)

// 画点
ctx.beginPath()
ctx.arc(100, 75, 2, 0, 2 * Math.PI)
ctx.setFillStyle('lightgreen')
ctx.fill()

ctx.beginPath()
ctx.arc(100, 25, 2, 0, 2 * Math.PI)
ctx.setFillStyle('blue')
ctx.fill()

ctx.beginPath()
ctx.arc(150, 75, 2, 0, 2 * Math.PI)
ctx.setFillStyle('red')
ctx.fill()

// 画弧线
ctx.beginPath()
ctx.arc(100, 75, 50, 0, 1.5 * Math.PI)
ctx.setStrokeStyle('#333333')
ctx.stroke()

ctx.draw()
```

针对 arc（100, 75, 50, 0, 1.5 * Math.PI）的 3 个关键坐标如下：

- 圆心（100, 75）。
- 起始弧度（0）。
- 终止弧度（1.5 * Math.PI）。

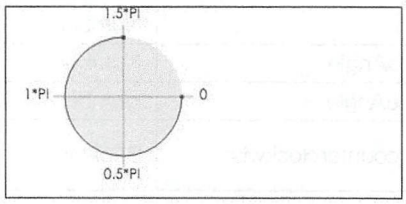


图 4-30 画弧线

4.8.7.8 quadraticCurveTo

quadraticCurveTo 用于创建二次贝塞尔曲线路径。quadraticCurveTo 的参数如表 4-215 所示。

表 4-215 quadraticCurveTo 的参数

参数	类型	说明
cpx	Number	贝塞尔控制点的 X 轴坐标
cpy	Number	贝塞尔控制点的 Y 轴坐标
x	Number	结束点的 X 轴坐标
y	Number	结束点的 Y 轴坐标

使用代码如下所示，显示效果如图 4-31 所示。

```
// 4.8.7.8  quadraticCurveTo
    const ctx = wx.createCanvasContext('testCanvas')

    // 画点
    ctx.beginPath()
    ctx.arc(20, 20, 2, 0, 2 * Math.PI)
    ctx.setFillStyle('red')
    ctx.fill()

    ctx.beginPath()
    ctx.arc(200, 20, 2, 0, 2 * Math.PI)
    ctx.setFillStyle('lightgreen')
    ctx.fill()

    ctx.beginPath()
    ctx.arc(20, 100, 2, 0, 2 * Math.PI)
    ctx.setFillStyle('blue')
    ctx.fill()

    ctx.setFillStyle('black')
    ctx.setFontSize(12)

    // 画导引线
    ctx.beginPath()
    ctx.moveTo(20, 20)
    ctx.lineTo(20, 100)
    ctx.lineTo(200, 20)
    ctx.setStrokeStyle('#AAAAAA')
    ctx.stroke()

    // 画二次贝塞尔曲线
    ctx.beginPath()
    ctx.moveTo(20, 20)
    ctx.quadraticCurveTo(20, 100, 200, 20)
    ctx.setStrokeStyle('black')
    ctx.stroke()

    ctx.draw()
```

针对 moveTo（20, 20）、quadraticCurveTo（20, 100, 200, 20）的 3 个关键坐标如下：

- 起始点（20, 20）。
- 控制点（20, 100）。
- 终止点（200, 20）。

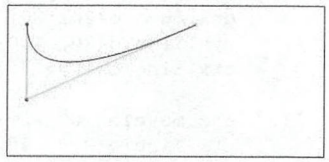


图 4-31 二次贝塞尔曲线

4.8.7.9 bezierCurveTo

bezierCurveTo 用于创建三次方贝塞尔曲线路径。bezierCurveTo 的参数如表 4-216 所示。

表 4-216 bezierCurveTo 的参数

参数	类型	说明
cp1x	Number	第一个贝塞尔控制点的 *X* 轴坐标
cp1y	Number	第一个贝塞尔控制点的 *Y* 轴坐标

续表

参数	类型	说明
cp2x	Number	第二个贝塞尔控制点的 X 轴坐标
cp2y	Number	第二个贝塞尔控制点的 Y 轴坐标
x	Number	结束点的 X 轴坐标
y	Number	结束点的 Y 轴坐标

使用代码如下所示，显示效果如图 4-32 所示。

```
// 4.8.7.9 bezierCurveTo
    const ctx = wx.createCanvasContext('testCanvas')

    // 画点
    ctx.beginPath()
    ctx.arc(20, 20, 2, 0, 2 * Math.PI)
    ctx.setFillStyle('red')
    ctx.fill()

    ctx.beginPath()
    ctx.arc(200, 20, 2, 0, 2 * Math.PI)
    ctx.setFillStyle('lightgreen')
    ctx.fill()

    ctx.beginPath()
    ctx.arc(20, 100, 2, 0, 2 * Math.PI)
    ctx.arc(200, 100, 2, 0, 2 * Math.PI)
    ctx.setFillStyle('blue')
    ctx.fill()

    ctx.setFillStyle('black')
    ctx.setFontSize(12)

    // 画导引线
    ctx.beginPath()
    ctx.moveTo(20, 20)
    ctx.lineTo(20, 100)
    ctx.lineTo(150, 75)

    ctx.moveTo(200, 20)
    ctx.lineTo(200, 100)
    ctx.lineTo(70, 75)
    ctx.setStrokeStyle('#AAAAAA')
    ctx.stroke()

    // 画三次方贝塞尔曲线
    ctx.beginPath()
    ctx.moveTo(20, 20)
    ctx.bezierCurveTo(20, 100, 200, 100, 200, 20)
    ctx.setStrokeStyle('black')
    ctx.stroke()

    ctx.draw()
```

针对 moveTo（20, 20）、bezierCurveTo（20, 100, 200, 100, 200, 20）的 3 个关键

坐标如下：

- 起始点（20, 20）。
- 两个控制点（20, 100）、（200, 100）。
- 终止点（200, 20）。

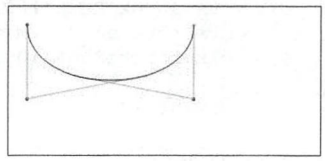


图 4-32 三次方贝塞尔曲线

4.8.8 变形

4.8.8.1 scale

在调用 scale 方法后创建的路径，其横纵坐标会被缩放。多次调用 scale，缩放的倍数会相乘。scale 的参数如表 4-217 所示。

表 4-217 scale 的参数

参数	类型	说明
scaleWidth	Number	横坐标缩放的倍数（1 = 100%；0.5 = 50%；2 = 200%）
scaleHeight	Number	纵坐标缩放的倍数（1 = 100%；0.5 = 50%；2 = 200%）

使用代码如下所示，显示效果如图 4-33 所示。

```
// 4.8.8.1 scale
    const ctx = wx.createCanvasContext('testCanvas')

    ctx.strokeRect(10, 10, 25, 15)
    ctx.scale(2, 2)
    ctx.strokeRect(10, 10, 25, 15)
    ctx.scale(2, 2)
    ctx.strokeRect(10, 10, 25, 15)

    ctx.draw()
```

4.8.8.2 rotate

rotate 以原点为中心顺时针旋转当前坐标轴，原点可以用 translate 方法修改。多次调用 rotate，旋转的角度会叠加。rotate 的参数如表 4-218 所示。

表 4-218 rotate 的参数

参数	类型	说明
rotate	Number	旋转角度，以弧度计算（degrees * Math.PI/180；degrees 范围为 0～360）

使用代码如下所示，显示效果如图 4-34 所示。

```
// 4.8.8.2 rotate
    const ctx = wx.createCanvasContext('testCanvas')

    ctx.strokeRect(100, 10, 150, 100)
    ctx.rotate(20 * Math.PI / 180)
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
ctx.strokeRect(100, 10, 150, 100)
ctx.rotate(20 * Math.PI / 180)
ctx.strokeRect(100, 10, 150, 100)

ctx.draw()
```

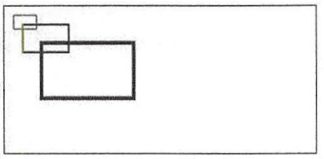


图 4-33 放大

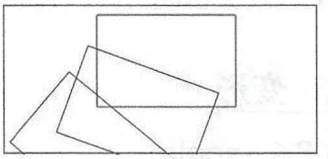


图 4-34 旋转

4.8.8.3 translate

translate 对当前坐标系的原点（0，0）进行变换，默认的坐标系原点为页面左上角。translate 的参数说明如表 4-219 所示。

表 4-219 translate 的参数说明

参数	类型	说明
x	Number	水平坐标平移量
y	Number	垂直坐标平移量

使用代码如下所示，显示效果如图 4-35 所示。

```
// 4.8.8.3 translate
    const ctx = wx.createCanvasContext('testCanvas')

    ctx.strokeRect(10, 10, 150, 100)
    ctx.translate(20, 20)
    ctx.strokeRect(10, 10, 150, 100)
    ctx.translate(20, 20)
    ctx.strokeRect(10, 10, 150, 100)

    ctx.draw()
```

4.8.9 文字（设置字号/绘制文本）

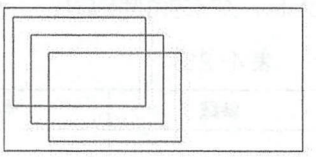


图 4-35 位移

setFontSize 用于设置字体的字号。setFontSize 的参数说明如表 4-220 所示。

表 4-220 setFontSize 的参数说明

参数	类型	说明
fontSize	Number	字体的字号

使用代码如下所示，显示效果如图 4-36 所示。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
//设置字号
    const ctx = wx.createCanvasContext('testCanvas')

    ctx.setFontSize(20)
    ctx.fillText('20', 20, 20)
    ctx.setFontSize(30)
    ctx.fillText('30', 40, 40)
    ctx.setFontSize(40)
    ctx.fillText('40', 60, 60)
    ctx.setFontSize(50)
    ctx.fillText('50', 90, 90)

    ctx.draw()
```

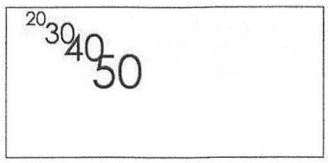


图 4-36 设置字号

fillText 用于在画布上绘制被填充的文本。fillText 的参数说明如表 4-221 所示。

表 4-221 fillText 的参数说明

参数	类型	说明
text	String	在画布上输出的文本
x	Number	绘制文本的左上角的 *X* 轴坐标位置
y	Number	绘制文本的左上角的 *Y* 轴坐标位置

使用代码如下所示，显示效果如图 4-37 所示。

```
//填充的文本
    const ctx = wx.createCanvasContext('testCanvas')

    ctx.setFontSize(20)
    ctx.fillText('Hello', 20, 20)
    ctx.fillText('MINA', 100, 100)

    ctx.draw()
```

setTextAlign 用于设置文字的对齐。从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持。setTextAlign 的参数说明如表 4-222 所示。

表 4-222 setTextAlign 的参数说明

参数	类型	说明
align	String	可选值 'left'、'center'、'right'

使用代码如下所示，显示效果如图 4-38 所示。

```
//文字对齐
    const ctx = wx.createCanvasContext('testCanvas')
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
ctx.setStrokeStyle('red')
ctx.moveTo(150, 20)
ctx.lineTo(150, 170)
ctx.stroke()

ctx.setFontSize(15)
ctx.setTextAlign('left')
ctx.fillText('textAlign=left', 150, 60)

ctx.setTextAlign('center')
ctx.fillText('textAlign=center', 150, 80)

ctx.setTextAlign('right')
ctx.fillText('textAlign=right', 150, 100)

ctx.draw()
```

图 4-37 填充文本

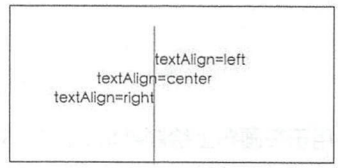


图 4-38 文字对齐

canvasContext.setTextBaseline 用于设置文字的水平对齐，从基础库 1.9.90 开始支持。语法如下所示：

```
canvasContext.setTextBaseline(textBaseline)
canvasContext.textBaseline = textBaseline // 从基础库 1.9.90 开始支持
```

参数 textBaseline 可以选择可选值 'top'、'bottom'、'middle'、'normal'。使用代码如下所示：

```
//文本上下对齐
const ctx = wx.createCanvasContext('testCanvas')

ctx.setStrokeStyle('red')
ctx.moveTo(5, 75)
ctx.lineTo(295, 75)
ctx.stroke()

ctx.setFontSize(20)

ctx.setTextBaseline('top')
ctx.fillText('top', 5, 75)

ctx.setTextBaseline('middle')
ctx.fillText('middle', 50, 75)

ctx.setTextBaseline('bottom')
ctx.fillText('bottom', 120, 75)

ctx.setTextBaseline('normal')
ctx.fillText('normal', 200, 75)
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
ctx.draw()
```

显示效果如图 4-39 所示。

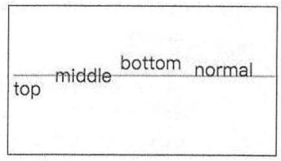


图 4-39 文本上下对齐效果

4.8.10 图片（drawImage）

drawImage 用于绘制图像，图像保持原始尺寸。drawImage 的参数说明如表 4-223 所示。

表 4-223 drawImage 的参数说明

参数	类型	说明
imageResource	String	所要绘制的图片资源
x	Number	图像左上角的 *X* 轴坐标
y	Number	图像左上角的 *Y* 轴坐标
width	Number	图像宽度
height	Number	图像高度

使用代码如下所示，显示效果如图 4-40 所示。

```
// 4.8.10 图片(drawImage)
    const ctx = wx.createCanvasContext('testCanvas')

    wx.chooseImage({
      success: function (res) {
        ctx.drawImage(res.tempFilePaths[0], 0, 0, 150, 100)
        ctx.draw()
      }
    })
```

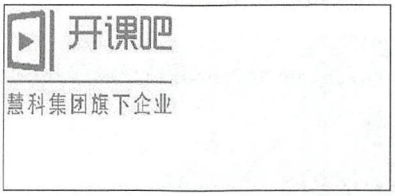


图 4-40 绘制图像

4.8.11 全局画笔透明度（setGlobalAlpha）

setGlobalAlpha 用于设置全局画笔透明度。setGlobalAlpha 的参数说明如表 4-224 所示。

表 4-224 setGlobalAlpha 的参数说明

参数	类型	范围	说明
alpha	Number	0～1	透明度，0 表示完全透明，1 表示完全不透明

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

使用代码如下所示，显示效果如图 4-41 所示。

```
// 4.8.11 全局画笔透明度(setGlobalAlpha)
    const ctx = wx.createCanvasContext('testCanvas')

    ctx.setFillStyle('red')
    ctx.fillRect(10, 10, 150, 100)
    ctx.setGlobalAlpha(0.2)
    ctx.setFillStyle('blue')
    ctx.fillRect(50, 50, 150, 100)
    ctx.setFillStyle('yellow')
    ctx.fillRect(100, 100, 150, 100)

    ctx.draw()
```

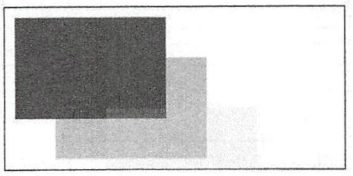


图 4-41 全局画笔透明度

4.8.12 其他

4.8.12.1 保存/恢复

save 保存当前的绘图上下文。

restore 恢复之前保存的绘图上下文。

使用代码如下所示，显示效果如图 4-42 所示。

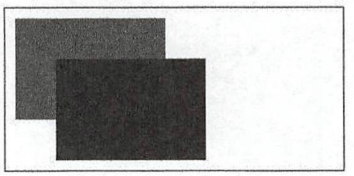


图 4-42 保存/恢复

```
// 4.8.12.1 保存/恢复
    const ctx = wx.createCanvasContext('testCanvas')

    // 保存默认的填充样式
    ctx.save()
    ctx.setFillStyle('red')
    ctx.fillRect(10, 10, 150, 100)

    // 恢复到以前保存的状态
    ctx.restore()
    ctx.fillRect(50, 50, 150, 100)

    ctx.draw()
```

4.8.12.2 draw

draw 将之前在绘图上下文中的描述（路径、变形、样式）画到 canvas 中。绘图上下文需要由 wx.createCanvasContext（canvasId）来创建。draw 的参数说明如表 4-225 所示。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 4-225　　draw 的参数说明

参数	类型	说明
reserve	Boolean	非必填。本次绘制是否接着上一次绘制，即 reserve 参数为 false，则在本次调用 drawCanvas 绘制之前，native 层应先清空画布再继续绘制；若 reserve 参数为 true，则保留当前画布上的内容，本次调用 drawCanvas 绘制的内容覆盖在上面。默认为 false

示例 1 代码如下所示，显示效果如图 4-43 所示。

```
// 4.8.12.2  draw
    const ctx = wx.createCanvasContext('testCanvas')

    ctx.setFillStyle('red')
    ctx.fillRect(10, 10, 150, 100)
    ctx.draw()
    ctx.fillRect(50, 50, 150, 100)
    ctx.draw()
```

示例 2 代码如下所示，显示效果如图 4-44 所示。

```
const ctx = wx.createCanvasContext('testCanvas')

    ctx.setFillStyle('red')
    ctx.fillRect(10, 10, 150, 100)
    ctx.draw()
    ctx.fillRect(50, 50, 150, 100)
    ctx.draw(true)
```

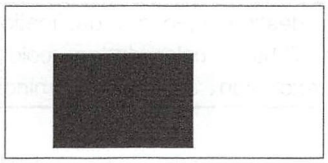


图 4-43　draw 方法，示例 1

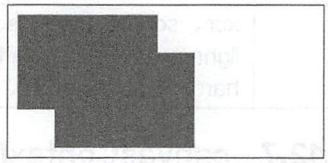


图 4-44　draw 方法，示例 2

4.8.12.3　getActions

getActions（不推荐使用），返回绘图上下文的绘图动作。

4.8.12.4　clearActions

clearActions（不推荐使用），清空绘图上下文的绘图动作。

4.8.12.5　canvasContext.measureText

测量文本尺寸信息，目前仅返回文本宽度，为同步接口。参数是要测量的文本字符串。返回 Number 类型的文本宽度。

使用代码如下所示：

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.font = 'italic bold 20px cursive'
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
const metrics = ctx.measureText('Hello World')
console.log(metrics.width)
```

ctx.font 用来设置字体样式和字号。

metrics 就是返回的字符串'Hello World'的尺寸信息，包括宽度和高度。

4.8.12.6 canvasContext.globalCompositeOperation

该属性设置在绘制新形状时使用的合成操作的类型。

语法如下所示：

```
canvasContext.globalCompositeOperation = type
```

globalCompositeOperation 的参数说明如下所表 4-226 所示。

表 4-226 globalCompositeOperation 的参数说明

参数	类型	说明
type	String	标识要使用哪种合成或混合模式操作

type 支持的操作，如表 4-227 所示。

表 4-227 type 支持的操作

平台	操作
安卓	xor、source-over、source-atop、destination-out、lighter、overlay、darken、lighten、hard-light
iOS	xor、source-over、source-atop、destination-over、destination-out、lighter、multiply、overlay、darken、lighten、color-dodge、color-burn、hard-light、soft-light、difference、exclusion、saturation、luminosity

4.8.12.7 canvasContext.arcTo

根据控制点和半径绘制圆弧路径。

语法如下所示：

```
canvasContext.arcTo(x1, y1, x2, y2, radius)
```

arcTo 的参数说明如表 4-228 所示。

表 4-228 arcTo 的参数说明

参数	类型	说明
x1	Number	第一个控制点的 *X* 轴坐标
y1	Number	第一个控制点的 *Y* 轴坐标
x2	Number	第二个控制点的 *X* 轴坐标
y2	Number	第二个控制点的 *Y* 轴坐标
radius	Number	圆弧的半径

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

4.8.12.8 canvasContext.strokeText

根据给定的（x, y）坐标绘制文本描边的方法。

语法如下所示：

```
canvasContext.strokeText(text, x, y, maxWidth)
```

strokeText 的参数说明如表 4-229 所示。

表 4-229 strokeText 的参数说明

参数	类型	说明
text	String	要绘制的文本
x	Number	文本起始点的 X 轴坐标
y	Number	文本起始点的 Y 轴坐标
maxWidth	Number	需要绘制的最大宽度，可选

4.8.12.9 canvasContext.lineDashOffset

设置虚线偏移量的属性。

语法如下所示：

```
canvasContext.lineDashOffset = value
```

lineDashOffset 的参数说明如表 4-230 所示。

表 4-230 lineDashOffset 的参数说明

参数	类型	说明
value	Number	偏移量，初始值为 0

4.8.12.10 canvasContext.createPattern

对指定的图像创建模式的方法，可在指定的方向上重复原图像。

语法如下所示：

```
canvasContext.createPattern(image, repetition)
```

createPattern 的参数说明如表 4-231 所示。

表 4-231 createPattern 的参数说明

参数	类型	说明
image	String	重复的图像源，仅支持包内路径和临时路径
repetition	String	指定如何重复图像，有效值有：repeat、repeat-x、repeat-y、no-repeat

使用代码如下所示：

非卖品，仅供非商业用途或交流学习使用

```
const ctx = wx.createCanvasContext('myCanvas')
const pattern = ctx.createPattern('/path/to/image', 'repeat-x')
ctx.fillStyle = pattern ctx.fillRect(0, 0, 300, 150)
ctx.draw()
```

4.8.12.11 font

设置当前字体样式的属性。

语法如下所示：

```
canvasContext.font = value
```

font 的参数说明如表 4-232 所示。

表 4-232 font 的参数说明

参数	类型	说明
value	String	符合 CSS font 语法的 DOMString 字符串，至少需要提供字体大小和字体族名。默认值为 10px sans-serif

其中 value 支持的参数如表 4-233 所示。

表 4-233 value 支持的参数

参数	说明
style	字体样式。仅支持 italic、oblique、normal
weight	字体粗细。仅支持 normal、bold
size	字体大小
family	字体族名。注意确认各平台所支持的字体

4.8.12.12 setTransform

使用矩阵重新设置（覆盖）当前变换的方法。

语法如下所示：

```
canvasContext.setTransform(scaleX, skewX, skewY, scaleY, translateX, translateY)
```

setTransform 的参数说明如表 4-234 所示。

表 4-234 setTransform 的参数说明

参数	类型	说明
scaleX	Number	水平缩放
skewX	Number	水平倾斜
skewY	Number	垂直倾斜
scaleY	Number	垂直缩放
translateX	Number	水平移动
translateY	Number	垂直移动

非卖品，仅供非商业用途或交流学习使用

4.9 拓展接口

wx.arrayBufferToBase64（arrayBuffer）：将 ArrayBuffer 数据转成 base64 字符串。从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持，具体代码如下所示：

```
const arrayBuffer = new Uint8Array([11, 22, 33]) const base64 = wx.arrayBufferToBase64 (arrayBuffer)
```

wx.base64ToArrayBuffer（base64）：将 base64 字符串转成 ArrayBuffer 数据。从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持。具体代码如下所示：

```
const base64 = 'CxYh' const arrayBuffer = wx.base64ToArrayBuffer(base64)
```

隐藏键盘 API：

```
wx.hideKeyboard()
```

停止下拉刷新动画 API：

```
wx.stopPullDownRefresh()
```

4.10 调试接口

从基础库版本 1.4.0 开始支持 wx.setEnableDebug(OBJECT)，该接口设置是否打开调试开关，此开关对正式版也能生效。

setEnableDebug 的 OBJECT 参数说明如表 4-235 所示。

表 4-235 setEnableDebug 的 OBJECT 参数说明

参数	类型	必填	说明
enableDebug	Boolean	是	是否打开调试
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-236 所示。

表 4-236 success 返回参数说明

参数	类型	说明
errMsg	String	调用结果

示例代码如下所示：

```
// 打开调试
wx.setEnableDebug({
```

非卖品，仅供非商业用途或交流学习使用

```
    enableDebug: true
  })

  // 关闭调试
  wx.setEnableDebug({
    enableDebug: false
  })
```

4.11 WXML 节点信息

1）wx.createSelectorQuery()

从基础库版本 1.4.0 开始支持，是获取一个节点信息的方法（createSelectorQuery 方法），该方法返回一个 selectorQuery 对象实例。在这个实例上可以使用 select 等方法选择节点，并使用 boundingClientRect 等方法选择需要查询的信息。

使用代码如下所示：

```
  <!--index.wxml-->
  <image src='{{imgUrl}}' class="book-image"  bindload="imageLoad" id="bookimageID">
</image>


  //index.js
  Page({
    data: {
      imgUrl: 'https://wx.leadingdo.com/images/wxBookIcon.jpg',
    },
    onLoad: function () {
      var _this = this;
      //创建节点选择器
      var query = wx.createSelectorQuery();
      //选择 id
      query.select('#bookimageID').boundingClientRect()
      query.selectViewport().scrollOffset()//这段代码的意思是获取页面滑动位置的查询请求
      query.exec(function (res) {
        //res 就是该元素的信息数组
        console.log('res',res);
        //取高度
        _this.setData({
                realWidth: res[0].width, // 获取宽度
                realHeight: res[0].height, // 获取高度
          scrollTop: res[1].scrollTop // 显示区域的竖直滚动位置
        })
        console.log('取高度', _this.data.realHeight);
        console.log('scrollTop:', _this.data.scrollTop);
      })
    },
    imageLoad:function(){

      console.log('imageLoad');
    }
  })
```

通过这两步开发者就可以获取一些相应的信息了，如图 4-45 所示。

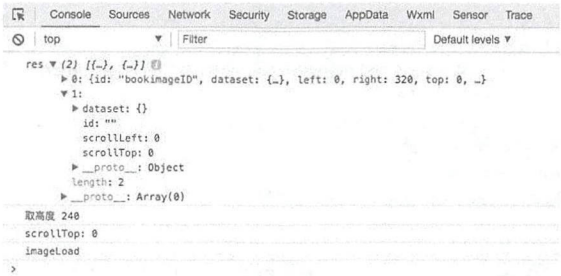


图 4-45 输出的节点信息

wx.createSelectorQuery()创建节点选择器，返回 selectorQuery 对象，selectorQuery 对象有以下方法可以使用，如表 4-237 所示。

表 4-237 selectorQuery 对象的方法列表

方法	参数	说明
in	object Component	参考下面详细介绍
select	selector	参考下面详细介绍
selectAll	selector	参考下面详细介绍
selectViewport		参考下面详细介绍
exec	[callback]	参考下面详细介绍

2）selectorQuery.in(component)

将选择器的选取范围更改为自定义组件 component 内。（初始时，选择器仅选取页面范围的节点，不会选取任何自定义组件中的节点。）

自定义组件简单地分为 3 个步骤，1. 创建组件；2. 编写组件；3. 调用组件。

第 1 步：创建组件。

创建一个 LMYAlertView 文件夹，右键单击文件夹选择“新建-Component”，成功之后，LMYAlertView 文件夹里包含 4 个文件，如图 4-46 所示。

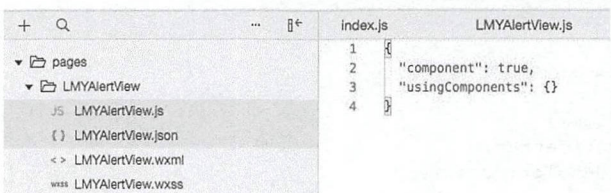


图 4-46 自定义组件 LMYAlertView

第 2 步：编写组件。

在 LMYAlertView.wxml 文件中输入以下代码：

```
<!--pages/LMYAlertView/LMYAlertView.wxml-->
<view hidden='{{alertHidden}}'>
  <view class='mask_layer' bindtap='alert_click_Hidden' />
  <view class='alert_box'>
  <image src='{{imgUrl}}' class="book-image" id="bookimageID"></image>
    <view class="title">提示</view>
    <view class='content'>
      <text class='alertMsg'>{{alertMsg}}</text>
    </view>
    <view class='btn'>
      <view bindtap='alert_click_Hidden' class='cancel'>取消</view>
      <view bindtap='Sure' class='Sure'>确定</view>
    </view>
  </view>
</view>
```

在 LMYAlertView.wxss 文件中输入以下代码：

```
    /* pages/LMYAlertView/LMYAlertView.wxss */
.mask_layer {
  width: 100%;
  height: 100%;
  position: fixed;
  z-index: 1000;
  background: #000;
  opacity: 0.5;
  overflow: hidden;
}

.book-image {
  width: 108rpx;
  height: 108rpx;
}

.alert_box {
  width: 76%;
  overflow: hidden;
  position: fixed;
  top: 50%;
  left: 0;
  z-index: 1001;
  background: #fafafa;
  margin: -150px 12% 0 12%;
  border-radius: 3px;
}

.title {
  padding: 15px;
  text-align: center;
  background-color: gazure;
}

.content {
  overflow-y: scroll; /*超出父盒子高度可滚动*/
}
```

```
.btn {
  width: 100%;
  margin-top: 65rpx;
  display: flex;
  flex-direction: row;
  align-items: center;
  justify-content: space-between;
  box-sizing: border-box;
  background-color: white;
}

.cancel {
  width: 100%;
  padding: 10px;
  text-align: center;
  color: red;
}

.Sure {
  width: 100%;
  padding: 10px;
  background-color: gainsboro;
  text-align: center;
}

.alertMsg {
  text-align: center;
  margin-top: 45rpx;
  display: block;
}
```

在 LMYAlertView.js 文件中输入以下代码：

```
    // pages/LMYAlertView/LMYAlertView.js
Component({
 /**
   * 组件的属性列表
   */
  properties: {
    alertHidden: {
      type: Boolean,
      value: true
    }, //这里定义了 alertHidden 属性，属性值可以在组件使用时指定，写法为 alertHidden
    alertMsg: {
      type: String,
      value: ' ',
    }
  },
  /**
   * 组件的初始数据
   */
  data: {
    // 这里是一些组件内部数据
    text: "text",
    imgUrl: 'https://wx.leadingdo.com/images/wxBookIcon.jpg',
  },
```

```
    methods: {
      // 这里放置自定义方法
      alert_click_Hidden: function () {
        this.setData({
          alertHidden: true,
        })
        console.log("您单击了取消按钮！")
      },
      // 确定
      Sure: function () {
        console.log("您单击了确定按钮！")

        var query = wx.createSelectorQuery().in(this)
        query.select('#bookimageID').boundingClientRect(function (res) {
                res.top // 这个组件内 #the-id 节点的上边界坐标
                console.log('节点的上边界坐标:', res.top);
        }).exec()
      },
      queryMultipleNodes: function () {

      }
    }
      })
```

第 3 步：调用组件。

在 pages/index/index 页面调用 pages/LMYAlertView/LMYAlertView 自定义组件。

首先在 index.json 中进行引用说明，这里设置自定义组件的标签名和引用路径，代码如下所示：

```
{
  "usingComponents": {
    "LMYAlertView": "../LMYAlertView/LMYAlertView"
  }
}
```

然后在 index.wxml 中调用组件，代码如下所示：

```
<!--index.wxml-->
<image src='{{imgUrl}}' class="book-image"  bindload="imageLoad" id="bookimageID">
</image>

<!-- 调用 modal 组件 -->
<LMYAlertView alert-hidden="{{is_alert_Hidden}}" alert-msg="{{is_alert_Msg}}"/>
```

在 index.js 中绑定要使用的数据，代码如下所示：

```
//index.js
Page({
  data: {
    alert_Hidden: false,
    is_alert_Msg: '这是一个自定义组件（弹框）',
  }
})
```

运行代码显示效果如图 4-47 所示。

在上面自定义组件中，可以看到使用了 selectorQuery.in 方法。单击确定按钮触发该方法。

3）selectorQuery.select(selector)

在当前页面下选择第一个匹配选择器 selector 的节点，返回一个 NodesRef 对象实例，可以用于获取节点信息。

selector 类似于 CSS 的选择器，但仅支持下列语法。

- ID 选择器：#the-id。
- class 选择器（可以连续指定多个）：.a-class.another-class。
- 子元素选择器：.the-parent > .the-child。
- 后代选择器：.the-ancestor .the-descendant。
- 跨自定义组件的后代选择器：.the-ancestor >>> .the-descendant。
- 多选择器的并集：#a-node, .some-other-nodes。

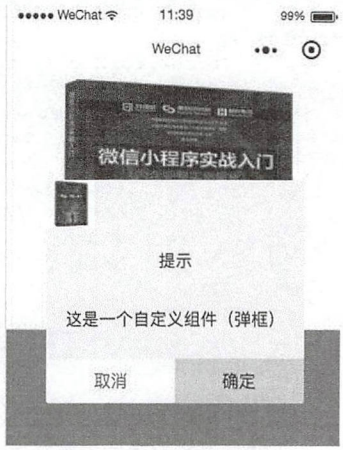


图 4-47 自定义组件

4）selectorQuery.selectAll(selector)

在当前页面下选择匹配选择器 selector 的节点，返回一个 NodesRef 对象实例。与 selectorQuery.selectNode(selector)不同的是，它选择所有匹配选择器的节点。

5）selectorQuery.selectViewport()

选择显示区域，可用于获取显示区域的尺寸、滚动位置等信息，返回一个 NodesRef 对象实例。

6）nodesRef.boundingClientRect([callback])

添加节点布局位置的查询请求，相对于显示区域，以像素为单位。其功能类似于 DOM 的 getBoundingClientRect。返回值是 nodesRef 对应的 selectorQuery。

返回的节点信息中，每个节点用 left、right、top、bottom、width、height 字段描述。如果提供了 callback 回调函数，在执行 selectQuery 的 exec 方法后，节点信息会在 callback 中返回。

使用代码如下所示：

```
//index.js
Page({
  getRect: function () {
    wx.createSelectorQuery().select('#the-id').boundingClientRect(function (rect) {
      rect.id      // 节点的 ID
      rect.dataset // 节点的 dataset
      rect.left    // 节点的左边界坐标
      rect.right   // 节点的右边界坐标
      rect.top     // 节点的上边界坐标
      rect.bottom  // 节点的下边界坐标
      rect.width   // 节点的宽度
      rect.height  // 节点的高度
    }).exec()
```

```
    },
    getAllRects: function () {
      wx.createSelectorQuery().selectAll('.a-class').boundingClientRect(function
(rects) {
        rects.forEach(function (rect) {
          rect.id      // 节点的 ID
          rect.dataset // 节点的 dataset
          rect.left    // 节点的左边界坐标
          rect.right   // 节点的右边界坐标
          rect.top     // 节点的上边界坐标
          rect.bottom  // 节点的下边界坐标
          rect.width   // 节点的宽度
          rect.height  // 节点的高度
        })
      }).exec()
    }
  });
```

7）nodesRef.scrollOffset([callback])

添加节点的滚动位置查询请求，以像素为单位。节点必须是 scroll-view 或者 viewport。返回值是 nodesRef 对应的 selectorQuery。

返回的节点信息中，每个节点的滚动位置用 scrollLeft、scrollTop 字段描述。如果提供了 callback 回调函数，在执行 selectQuery 的 exec 方法后，节点信息会在 callback 中返回。

使用代码如下所示：

```
Page({
  getScrollOffset: function () {
    wx.createSelectorQuery().selectViewport().scrollOffset(function (res) {
      res.id      // 节点的 ID
      res.dataset // 节点的 dataset
      res.scrollLeft // 节点的水平滚动位置
      res.scrollTop  // 节点的竖直滚动位置
    }).exec()
  }
})
```

8）nodesRef.fields(fields, [callback])

获取节点的相关信息，需要获取的字段在 fields 中指定。返回值是 nodesRef 对应的 selectorQuery。可指定获取的字段如表 4-238 所示。

表 4-238　　指定获取的字段

字段名	默认值	说明
id	否	是否返回节点 id
dataset	否	是否返回节点 dataset
rect	否	是否返回节点布局位置（left、right、top、bottom）
size	否	是否返回节点尺寸（width、height）
scrollOffset	否	是否返回节点的 scrollLeft、scrollTop，节点必须是 scroll-view 或者 viewport

续表

字段名	默认值	说明
properties	[]	指定属性名列表，返回节点对应属性名的当前属性值（只能获得组件文档中标注的常规属性值， id class style 和事件绑定的属性值不可获取）

使用代码如下所示：

```
Page({
  getFields: function () {
    wx.createSelectorQuery().select('#the-id').fields({
      dataset: true,
      size: true,
      scrollOffset: true,
      properties: ['scrollX', 'scrollY']
    }, function (res) {
      res.dataset    // 节点的 dataset
      res.width      // 节点的宽度
      res.height     // 节点的高度
      res.scrollLeft // 节点的水平滚动位置
      res.scrollTop  // 节点的竖直滚动位置
      res.scrollX    // 节点 scroll-x 属性的当前值
      res.scrollY    // 节点 scroll-y 属性的当前值
    }).exec()
  }
})
```

9）selectorQuery.exec([callback])

执行所有的请求，请求结果按请求次序构成数组，在 callback 的第一个参数中返回。

4.12 WXML 节点布局相交状态

节点布局交叉状态 API 可用于监听两个或多个组件节点在布局位置上的相交状态。这一组 API 常常可以用于推断某些节点是否可以被用户看见、有多大比例可以被用户看见。

这一组 API 涉及的主要概念如下：

- 参照节点：监听的参照节点，取它的布局区域作为参照区域。如果有多个参照节点，则取它们布局区域的交集作为参照区域。页面显示区域也可作为参照区域之一。
- 目标节点：监听的目标，只能是一个节点。
- 相交区域：目标节点的布局区域与参照区域的相交区域。
- 相交比例：相交区域占参照区域的比例。
- 阈值：相交比例如果达到阈值，则会触发监听器的回调函数。阈值可以有多个。

以下示例代码可以在目标节点（用选择器 .target-class 指定）每次进入或离开页面显示区域时，触发回调函数。

使用代码如下所示：

```
Page({
  onLoad: function () {
    wx.createIntersectionObserver().relativeToViewport().observe('.target-
class', (res) => {
      res.intersectionRatio // 相交区域占目标节点的布局区域的比例
      res.intersectionRect // 相交区域
      res.intersectionRect.left // 相交区域的左边界坐标
      res.intersectionRect.top // 相交区域的上边界坐标
      res.intersectionRect.width // 相交区域的宽度
      res.intersectionRect.height // 相交区域的高度
    })
  }
})
```

以下示例代码可以在目标节点（用选择器 .target-class 指定）与参照节点（用选择器 .relative-class 指定）在页面显示区域内相交或相离，且相交或相离程度达到目标节点布局区域的 20%和 50%时，触发回调函数。使用代码如下所示：

```
Page({
  onLoad: function () {
    wx.createIntersectionObserver({
      thresholds: [0.2, 0.5]
    }).relativeTo('.relative-class').relativeToViewport().observe('.target-
class', (res) => {
      res.intersectionRatio // 相交区域占目标节点的布局区域的比例
      res.intersectionRect // 相交区域
      res.intersectionRect.left // 相交区域的左边界坐标
      res.intersectionRect.top // 相交区域的上边界坐标
      res.intersectionRect.width // 相交区域的宽度
      res.intersectionRect.height // 相交区域的高度
    })
  }
})
```

1）wx.createIntersectionObserver([this], [options])

从基础库版本 1.9.3 开始支持，创建并返回一个 IntersectionObserver 对象实例。在自定义组件中，可以使用 this.createIntersectionObserver([options])来代替。

可选的 options 如表 4-239 所示。

表 4-239　　可选的 options

字段名	类型	说明
Thresholds	Array	一个数值数组，包含所有阈值。默认为 [0]
initialRatio	Number	初始的相交比例，如果调用时检测到的相交比例与这个值不相等且达到阈值，则会触发一次监听器的回调函数。默认为 0

2）intersectionObserver.relativeTo(selector, [margins])

使用选择器指定一个节点作为参照区域之一。margins 可以用来扩展（或收缩）参照节点布局区域的边界，可包含 left、right、top、bottom 4 项。

3）intersectionObserver.relativeToViewport([margins])

指定页面显示区域作为参照区域之一。margins 可以用来扩展（或收缩）参照节点布局区域的边界，可包含 left、right、top、bottom 4 项。

下面的示例代码中，如果目标节点（用选择器 .target-class 指定）进入显示区域以下 100px 时，就会触发回调函数。

```
Page({
  onLoad: function () {
    wx.createIntersectionObserver().relativeToViewport({ bottom: 100 }).
observe('.target-class', (res) => {
      res.intersectionRatio // 相交区域占目标节点的布局区域的比例
      res.intersectionRect // 相交区域
      res.intersectionRect.left // 相交区域的左边界坐标
      res.intersectionRect.top // 相交区域的上边界坐标
      res.intersectionRect.width // 相交区域的宽度
      res.intersectionRect.height // 相交区域的高度
    })
  }
})
```

4）intersectionObserver.observe(targetSelector, callback)

指定目标节点并开始监听相交状态变化情况。回调函数 callback 包含一个参数 result 。

回调函数 result 包含的字段如表 2-240 所示。

表 2-240　　回调函数 result 包含的字段

字段名	类型	说明
intersectionRatio	Number	相交比例
intersectionRect	Object	相交区域的边界，包含 left、right、top、bottom 4 项
boundingClientRect	Object	目标节点布局区域的边界，包含 left、right、top、bottom 4 项
relativeRect	Object	参照区域的边界，包含 left、right、top、bottom 4 项
time	Number	相交检测时的时间戳

5）intersectionObserver.disconnect()

停止监听。回调函数将不再触发。

提示：

与页面显示区域的相交区域并不准确代表用户可见的区域，因为参与计算的区域是“布局区域”，布局区域可能会在绘制时被其他节点裁剪隐藏（如节点中 overflow 样式为 hidden 的节点）或遮盖（如 fixed 定位的节点）。

4.13　第三方平台

1）wx.getExtConfig(OBJECT)

从基础库版本 1.1.0 开始支持，获取第三方平台自定义的数据字段。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

getExtConfig 的 OBJECT 参数说明如表 4-241 所示。

表 4-241 getExtConfig 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	返回第三方平台自定义的数据
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 4-242 所示。

表 4-242 success 返回参数说明

参数	类型	说明
errMsg	String	调用结果
extConfig	Object	第三方平台自定义的数据

wx.getExtConfig 暂时无法通过 wx.canIUse 判断是否兼容，开发者需要自行判断 wx.getExtConfig 是否兼容。

示例代码如下所示：

```
if (wx.getExtConfig) {
  wx.getExtConfig({
    success: function (res) {
      console.log(res.extConfig)
    }
  })
}
```

2）wx.getExtConfigSync()

从基础库版本 1.1.0 开始支持，获取第三方平台自定义的数据字段的同步接口。

返回参数说明如表 4-243 所示。

表 4-243 返回参数说明

参数	类型	说明
extConfig	Object	第三方平台自定义的数据

wx.getExtConfigSync 暂时无法通过 wx.canIUse 判断是否兼容，开发者需要自行判断 wx.getExtConfigSync 是否兼容。

示例代码如下所示：

```
let extConfig = wx.getExtConfigSync? wx.getExtConfigSync(): {} console.log
(extConfig)
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

第五章

开放 API

开放 API 主要包括调用微信登录、授权、获取用户登录信息、使用微信支付、模板消息、客服消息、转发分享、获取二维码、收货地址、卡卷、设置、微信运动、打开小程序。本章将详细做介绍。

5.1 登录

1）wx.login（OBJECT）：调用接口获取登录凭证（code），进而换取用户登录态信息，包括用户的唯一标识（openid）及本次登录的会话密钥（session_key）。用户数据的加密、解密通信需要依赖会话密钥完成。login 的 OBJECT 参数说明如表 5-1 所示。

表 5-1　　login 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 5-2 所示。

表 5-2　　success 返回参数说明

参数	类型	说明
errMsg	String	调用结果
code	String	用户允许登录后，回调内容会带上 code（有效期 5 分钟），开发者需要将 code 发送到开发者服务器后台，使用 code 换取 session_key API，将 code 换成 openid 和 session_key

新创建一个项目后，app.js 文件默认带有微信登录 API，代码如下所示：

```
//app.js
App({
  onLaunch: function () {
    //调用 API 从本地缓存中获取数据
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    var logs = wx.getStorageSync('logs') || []
    logs.unshift(Date.now())
    wx.setStorageSync('logs', logs)
  },
  getUserInfo: function (cb) {
    var that = this
    if (this.globalData.userInfo) {
      typeof cb == "function" && cb(this.globalData.userInfo)
    } else {
      //调用登录接口
      wx.login({
        success: function (res) {

          //登录成功


          //获取用户信息
          wx.getUserInfo({
            success: function (res) {
              that.globalData.userInfo = res.userInfo
              typeof cb == "function" && cb(that.globalData.userInfo)
            }
          })
        },
        fail: function (res) {
          console.log("微信登录失败")
        },
        complete: function (res) {
          console.log("微信登录完成")
        }
      })
    }
  },
  globalData: {
    userInfo: null
  }
})
```

wx.login 回调 success，可以从 success 返回的参数中获取 code 和 errMsg，参数返回内容如图 5-1 所示。

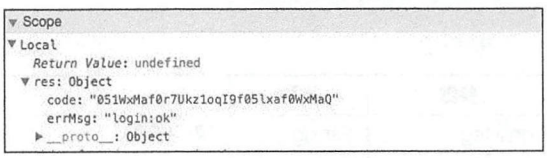


图 5-1 success 返回的参数

使用获取到的 code，调用下面接口，就可以获取 session_key 和 openid。session_key 是对用户数据进行加密签名的密钥。为了自身应用安全，session_key 不应该在网络上传输，而需要在微信公众平台服务器域名管理中增加“api.weixin.qq.com”才可以使用该接口。

```
    https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&j
s_code=JSCODE&grant_type=authorization_code
```

api.weixin.qq.com 用来获取 openid 和 session_key，请求参数说明如表 5-3 所示。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 5-3　　api.weixin.qq.com 的请求参数说明

参数	必填	说明
appid	是	小程序唯一标识
secret	是	小程序的 app secret
js_code	是	登录时获取的 code
grant_type	是	填写为 authorization_code

调用成功之后，返回参数说明如表 5-4 所示。

表 5-4　　成功返回的参数说明

参数	说明	参数
openid	用户唯一标识	openid

具体代码如下所示：

```
    //调用登录接口
        wx.login({
          success: function (res) {

            //登录成功——获取 openid 和 session_key

            //填写微信小程序 appid
            var appid = 'wx3602ec25ae74cfea';

            //填写微信小程序 secret
            var secret = '79361cb38609ae761b5b7861c4587781';

            var code = res.code;

            var urlTemp = 'https://api.weixin.qq.com/sns/jscode2session?appid='
+ appid + '&secret=' + secret + '&js_code=' + code + '&grant_type=
authorization_code';

            console.log("urlTemp=" + urlTemp);

            //调用 request 请求 api 转换登录凭证
            wx.request({
              url: urlTemp,
              header: {
                'content-type': 'application/json'
              },
              success: function (res) {

                //获取 openid
                console.log("openid = " + res.data.openid)

                //session_key
                console.log("session_key = " + res.data.session_key)
              },
              fail: function (res) {
                console.log("获取 openid 失败")
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
            },
            complete: function (res) {
              console.log("获取 openid 完成")
            }
          })

          //获取用户信息
          wx.getUserInfo({
            success: function (res) {
              that.globalData.userInfo = res.userInfo
              typeof cb == "function" && cb(that.globalData.userInfo)
            }
          })
        },
        fail: function (res) {
          console.log("微信登录失败")
        },
        complete: function (res) {
          console.log("微信登录完成")
        }
      })
    }
```

正常返回的 JSON 数据包：

```
{
   "session_key":"RoH2FTSZAa3aEucW0hU+Nw==",
   "expires_in":7200,
"openid":"oQ-Ds0BRTFiIGGnXm-dLCj1Rpge4"
}
```

错误时返回的 JSON 数据包（示例为 Code 无效）：

```
{
"errcode":40029,
"errmsg":"invalid code, hints: [ req_id: W0567ns80 ]"
}
```

2）wx.checkSession（OBJECT）：小程序通过上述接口获得的用户登录态拥有一定的时效性。用户越久未使用小程序，用户登录态越有可能失效。反之，如果用户一直在使用小程序，则用户登录态一直保持有效。具体时效逻辑由微信维护，不透明。开发者只需要调用 wx.checkSession 接口检测当前用户登录态是否有效。登录态过期后开发者可以再调用 wx.login 获取新的用户登录态。checkSession 的 OBJECT 参数说明如表 5-5 所示。

表 5-5　　checkSession 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数，登录态未过期
fail	Function	否	接口调用失败的回调函数，登录态已过期
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

使用代码如下所示：

```
//检测当前用户登录态是否有效
        wx.checkSession({
          success: function () {
            console.log("session 未过期，并且在本生命周期一直有效")
          },
          fail: function () {
            console.log("登录态过期——重新登录")
            wx.login()
          }
        })
```

3）登录态维护

小程序通过 wx.login() 获取到用户登录态之后，需要维护登录态。开发者要注意不应该直接把 session_key、openid 等字段作为用户的标识或者 session 的标识，而应该自己派发一个 session 登录态（如图 5-2 所示）。对于开发者自己生成的 session，应该保证其安全性且不应该设置较长的过期时间。session 派发到小程序客户端之后，可将其存储在 storage，用于后续通信使用。

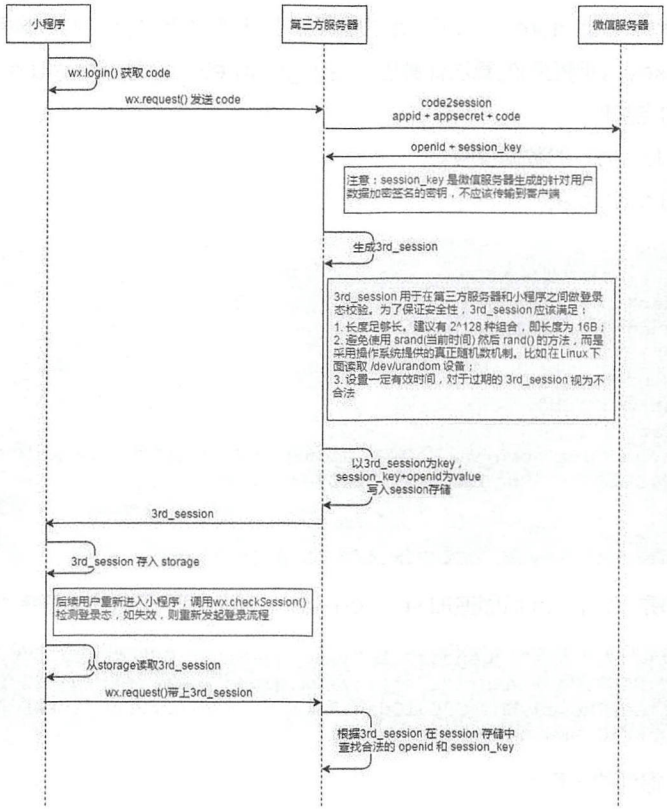


图 5-2 登录时序图

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

开发者可以通过 wx.checkSession() 检测用户登录态是否失效，并决定是否调用 wx.login() 重新获取登录态。

5.2 签名加密

本节主要介绍用户数据的签名验证和加解密。为了确保开放接口返回用户数据的安全性，微信会对明文数据进行签名。开发者可以根据业务需要对数据包进行签名校验，确保数据的完整性。

1）数据签名校验

数据签名校验步骤：

（1）签名校验算法涉及用户的 session_key，通过 wx.login 登录流程获取用户的 session_key，并自行维护与应用自身登录态的对应关系。

（2）通过调用接口（如 wx.getUserInfo）获取数据时，接口会同时返回 rawData、signature，其中 signature = sha1（rawData + session_key）。

（3）开发者将 signature、rawData 发送到开发者服务器进行校验。服务器利用用户对应的 session_key 使用相同的算法计算出签名 signature2，比对 signature 与 signature2 即可校验数据的完整性。

如 wx.getUserInfo 的数据校验。

接口返回的 rawData：

```
    {
        "nickName": "刘明洋",
        "gender": 1,
        "language": "zh_CN",
        "city": "",
        "province": "",
        "country": "CN",
        "avatarUrl":
"http://wx.qlogo.cn/mmopen/vi_32/DYAIOgq98piacDvPtg6PPRQyiapAcfCMovevyWMj4C9E
JTQdic17YmlzjlLFEJL5xtGKocT3vFWFo6lYywZBJtplg/0"
    }

    //用户的 session-key 是: qGfRUtxOb/UjOXMN6ZdeVw==
```

签名字符串是 rawData 后面追加 session-key，所以用于签名的字符串为：

```
    {"nickName":"刘明洋","gender":1,"language":"zh_CN","city":"", "province":
"", "country":"CN","avatarUrl":"http://wx.qlogo.cn/mmopen/vi_32/DYAIOgq98piac
DvPtg6PPRQyiapAcfCMovevyWMj4C9EJTQdic17YmlzjlLFEJL5xtGKocT3vFWFo6lYywZBJtplg
/0"} qGfRUtxOb/UjOXMN6ZdeVw==
```

使用 sha1 得到的结果为：

```
    44cd87d77b0bc42e8ff803a67d36ec48b1faca13
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

2）加密数据解密算法

如果接口涉及敏感数据（如 wx.getUserInfo 当中的 openId 和 unionId），接口的明文内容将不包含这些敏感数据。开发者如需要获取敏感数据，需要对接口返回的加密数据（encryptedData）进行对称解密。

解密算法如下：

（1）对称解密使用的算法为 AES-128-CBC，数据采用 PKCS#7 填充。

（2）对称解密的目标密文为 Base64_Decode（encryptedData）。

（3）对称解密密钥 aeskey = Base64_Decode（session_key），aeskey 是16字节。

（4）对称解密算法初始向量为 Base64_Decode（iv），其中 iv 由数据接口返回。

微信官方提供了多种编程语言的示例代码（单击 https://mp.weixin.qq.com/debug/wxadoc/dev/demo/aes-sample.zip 下载）。每种语言类型的接口名字均一致。调用方式可以参照示例。

另外，为了应用能校验数据的有效性，我们会给敏感数据加上数据水印（watermark），watermark 的参数说明如表 5-6 所示。

表 5-6　　watermark 的参数说明

参数	类型	说明
watermark	OBJECT	数据水印
appid	String	敏感数据归属 appid，开发者可校验此参数与自身 appid 是否一致
timestamp	DateInt	敏感数据获取的时间戳，开发者可以用于数据时效性校验

参考示例代码对 encryptedData 进行解密，解密之后获取 watermark 的值，获取到的信息类似于：

```
{
    "openId": "OPENID",
    "nickName": "NICKNAME",
    "gender": GENDER,
    "city": "CITY",
    "province": "PROVINCE",
    "country": "COUNTRY",
    "avatarUrl": "AVATARURL",
    "unionId": "UNIONID",
    "watermark":
    {
        "appid":"APPID",
        "timestamp":TIMESTAMP
    }
}
```

此前提供的加密数据（encryptData）以及对应的加密算法将被弃用，请开发者不要再依赖旧逻辑。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

5.3 授权

小程序从基础库版本 1.2.0 开始支持 wx.authorize(OBJECT) API，申请授权接口。

部分接口需要获得同意后才能调用。此类接口调用时，如果用户未授权过，会弹窗询问用户，用户单击同意后方可调用该接口。如果用户单击了拒绝，则短期内调用不会出现弹窗，而是直接进入 fail 回调。用户可以在小程序设置界面中修改对该小程序的授权信息。本接口用于提前向用户发起授权，调用后会立刻弹窗询问用户是否同意小程序使用某项功能或获取用户的某些数据，但不会实际调用接口。如果用户之前已经同意，则不会出现弹窗，直接返回成功。

authorize 的 OBJECT 参数说明如表 5-7 所示。

表 5-7　　authorize 的 OBJECT 参数说明

参数	类型	必填	说明
scope	String	是	需要获取权限的 scope，详见 scope 列表
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回 errMsg 参数，开发者可以通过该参数查看调用结果。

authorize 使用代码如下所示。

```
//开发者可以通过 wx.getSetting 先查询一下用户是否授权了 "scope.record" 这个 scope
wx.getSetting({
  success(res) {
    if (!res.authSetting['scope.record']) {
      wx.authorize({
        scope: 'scope.record',
        success() {
          // 用户已经同意小程序使用录音功能，后续调用 wx.startRecord 接口不会弹窗询问
          wx.startRecord()
        }
      })
    }
  }
})
```

scope 列表如表 5-8 所示。

表 5-8　　scope 列表

scope	对应接口	描述
scope.userInfo	wx.getUserInfo	用户信息
scope.userLocation	wx.getLocation、wx.chooseLocation	地理位置
scope.address	wx.chooseAddress	通讯地址

非卖品，仅供非商业用途或交流学习使用

续表

scope	对应接口	描述
scope.record	wx.startRecord	录音功能
scope.writePhotosAlbum	wx.saveImageToPhotosAlbum、wx.saveVideoToPhotosAlbum	保存到相册

5.4 用户信息

wx.getUserInfo（OBJECT）：使用该接口将不再出现授权弹窗，小程序使用 <button open-type="getUserInfo"></button>引导用户主动进行授权操作。如果用户未授权过，调用该接口将直接报错；如果用户已授权，可以使用该接口获取用户信息。getUserInfo 的 OBJECT 参数说明如表 5-9 所示。

表 5-9 getUserInfo 的 OBJECT 参数说明

参数	类型	必填	说明
withCredentials	Boolean	否	是否带上登录态信息
lang	String	否	指定返回用户信息的语言，zh_CN 为简体中文，en 为英文
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

withCredentials 字段从基础库版本 1.1.0 开始支持，低版本需做兼容处理。当 withCredentials 为 true 时，要求此前调用过 wx.login 且登录态尚未过期，此时返回的数据会包含 encryptedData、iv 等敏感信息；当 withCredentials 为 false 时，不要求有登录态，返回的数据不包含 encryptedData、iv 等敏感信息。

success 返回参数说明如表 5-10 所示。

表 5-10 success 返回参数说明

参数	类型	说明
userInfo	OBJECT	用户信息对象，不包含 openid 等敏感信息
rawData	String	不包括敏感信息的原始数据字符串，用于计算签名
signature	String	使用 sha1(rawData + session_key) 得到字符串，用于校验用户信息
encryptedData	String	包括敏感数据在内的完整用户信息的加密数据，详见加密数据解密算法
iv	String	加密算法的初始向量，详见加密数据解密算法

首次运行小程序，会弹出微信授权对话框，如图 5-3 所示。

userInfo 的参数说明如表 5-11 所示。

非卖品，仅供非商业用途或交流学习使用

表 5-11　　userInfo 的参数说明

参数	类型	说明
nickName	String	用户昵称
avatarUrl	String	用户头像，最后一个数值代表正方形头像大小（可选值有 0、46、64、96、132，0 代表 132×132 正方形头像），用户没有头像时该项为空。若用户更换头像，原有头像 URL 将失效
gender	String	用户的性别，值为 1 时是男性，值为 2 时是女性，值为 0 时是未知
city	String	用户所在城市
province	String	用户所在省份
country	String	用户所在国家
language	String	用户的语言，简体中文为 zh_CN

示例代码，如下所示：

```
<!--index.wxml-->
<!-- 如果只是展示用户头像昵称，可以使用<open-data />组件 -->
<open-data type="userAvatarUrl"></open-data>
<open-data type="userNickName"></open-data>
<!-- 需要使用 button 来授权登录 -->
<button wx:if="{{canIUse}}" open-type="getUserInfo" bindgetuserinfo=
"bindGetUserInfo">授权登录</button>
<view wx:else>请升级微信版本</view>

//index.js
Page({
  data: {
    canIUse: wx.canIUse('button.open-type.getUserInfo')
  },
  onLoad: function () {
    // 查看是否授权
    wx.getSetting({
      success: function (res) {
        if (res.authSetting['scope.userInfo']) {
          // 已经授权，可以直接调用 getUserInfo 获取头像昵称
          wx.getUserInfo({
            success: function (res) {
              console.log(res.userInfo)
            }
          })
        }
      }
    })
  },
  bindGetUserInfo: function (e) {
    console.log(e.detail.userInfo)
  }
})
```

UnionID 机制说明：

如果开发者拥有多个移动应用、网站应用和公众账号（包括小程序），可通过 unionid 来

非卖品，仅供非商业用途或交流学习使用

区分用户的唯一性，因为只要是同一个微信开放平台账号下的移动应用、网站应用和公众账号（包括小程序），用户的 unionid 是唯一的。换句话说，同一用户，对同一个微信开放平台下的不同应用，unionid 是相同的。

微信开放平台绑定小程序流程，前提是微信开放平台账号必须已完成开发者资质认证。

1）开发者资质认证流程

登录微信开放平台 — 账号中心 — 开发者资质认证，如图 5-4 所示。

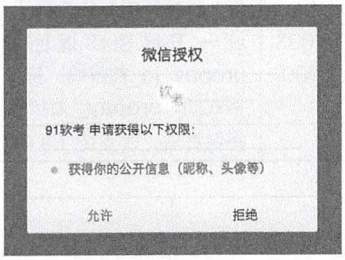


图 5-3 微信授权

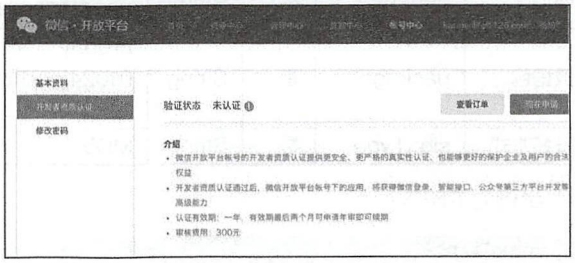


图 5-4 申请认证

2）绑定流程

登录微信开放平台 — 管理中心 — 公众账号 — 绑定公众账号，如图 5-5 所示。

图 5-5 绑定公众账号

5.5 微信支付

小程序调起支付数据签名字段列表，如表 5-12 所示。

非卖品，仅供非商业用途或交流学习使用

表 5-12　　数据签名字段

字段名	变量名	必填	类型	示例值	描述
小程序 ID	appId	是	String	wxd678efh567hg6787	微信分配的小程序 ID
时间戳	timeStamp	是	String	1490840662	时间戳从 1970 年 1 月 1 日 00:00:00 至今的秒数，即当前的时间
随机串	nonceStr	是	String	5K8264ILTKCH16CQ2502SI8ZNMTM67VS	随机字符串，不长于 32 位。推荐随机数生成算法
数据包	package	是	String	prepay_id=wx2017033010242291fcfe0db70013231072	统一下单接口返回的 prepay_id 参数值，提交格式如：prepay_ id=*
签名方式	signType	是	String	MD5	签名算法，暂支持 MD5

举例如下所示：

```
    paySign =
    MD5(appId=wxd678efh567hg6787&nonceStr=5K8264ILTKCH16CQ2502SI8ZNMTM67VS&packag
e=prepay_id=wx2017033010242291fcfe0db70013231072&signType=MD5&timeStamp=1490840662
&key=qazwsxedcrfvtgbyhnujmikolp111111) = 22D9B4E54AB1950F51E0649E8810ACD6
```

wx.requestPayment（OBJECT）：发起微信支付。requestPayment OBJECT 参数说明如表 5-13 所示。

表 5-13　　requestPayment 的 OBJECT 参数说明

参数	类型	必填	说明
timeStamp	String	是	时间戳从 1970 年 1 月 1 日 00:00:00 至今的秒数，即当前的时间
nonceStr	String	是	随机字符串，长度为 32 个字符以下
package	String	是	统一下单接口返回的 prepay_id 参数值，提交格式如：prepay_id=*
signType	String	是	签名算法，暂支持 MD5
paySign	String	是	签名，具体签名方案参见微信公众号支付帮助文档
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

requestPayment 的回调结果如表 5-14 所示。

表 5-14　　requestPayment 的回调结果

回调类型	errMsg	说明
success	requestPayment:ok	调用支付成功
fail	requestPayment:fail cancel	用户取消支付
fail	requestPayment:fail（detail message）	调用支付失败，其中 detail message 为后台返回的详细失败原因

实例代码如下所示：

```
wx.requestPayment(
{
'timeStamp': '',
'nonceStr': '',
'package': '',
'signType': 'MD5',
'paySign': '',
'success':function(res){},
'fail':function(res){},
'complete':function(res){}
})
```

5.6 模板消息

基于微信的通知渠道，微信开发者提供了可以高效触达用户的模板消息能力，以便实现服务的闭环并提供更佳的体验。

模板推送位置：服务通知。

模板下发条件：用户本人在微信体系内与页面有交互行为后触发。

模板跳转能力：单击查看详情仅能跳转下发模板的该账号的各个页面。

5.6.1 使用说明

登录微信公众平台获取模板，如果没有合适的模板，可以申请添加新模板，审核通过后可使用。

可以从“模板消息-模板库”中选择需要的模板，如图 5-6 所示。

对于一个模板，开发者可以选择自己需要的属性，如图 5-7 所示。

图 5-6 模板消息

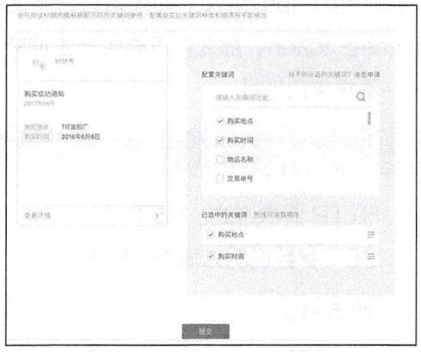


图 5-7 配置模板消息

页面的<form>组件，属性 report-submit 为 true 时，可以声明为需发模板消息，此时单击按钮提交表单可以获取 formId，用于发送模板消息。或者当用户完成支付行为，开发者可

以获取 prepay_id 用于发送模板消息。

5.6.2 接口说明

5.6.2.1 获取 access_token

access_token 是全局唯一接口调用凭据，开发者调用各接口时都需使用 access_token，请妥善保存。access_token 的存储至少要保留 512 个字符空间。access_token 的有效期目前为 2 小时，需定时刷新，重复获取将导致上次获取的 access_token 失效。

公众平台的 API 调用所需的 access_token 的使用及生成方式说明：

- 为了保密 appsecrect，第三方需要一个 access_token 获取和刷新的中控服务器。而其他业务逻辑服务器所使用的 access_token 均来自于该中控服务器，不应该各自去刷新，否则会造成 access_token 覆盖而影响业务；
- 目前 access_token 的有效期通过返回的 expires_in 来传达，目前是 7200s 之内的值。中控服务器需要根据这个有效时间提前去刷新 access_token。在刷新过程中，中控服务器对外输出的依然是老 access_token，此时公众平台后台会保证在刷新短时间内，新老 access_token 都可用，这保证了第三方业务的平滑过渡；
- access_token 的有效时间可能会在未来有调整，所以中控服务器不仅需要内部定时主动刷新，还需要提供被动刷新 access_token 的接口，这样便于业务服务器在 API 调用获知 access_token 已超时的情况下，可以触发 access_token 的刷新流程。

开发者可以使用 AppID 和 AppSecret 调用本接口来获取 access_token。AppID 和 AppSecret 可登录微信公众平台官网-设置-开发设置中获得。AppSecret 生成后请自行保存，因为在公众平台每次生成查看都会导致 AppSecret 被重置。注意调用所有微信接口时均需使用 https 协议。如果第三方不使用中控服务器，而是选择各个业务逻辑点各自去刷新 access_token，那么就可能会产生冲突，导致服务不稳定。

请求接口地址：

```
https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=APPID&secret=APPSECRET
```

HTTP 请求方式：GET。

GET 参数说明如表 5-15 所示。

表 5-15　　GET 参数说明

参数	必填	说明
grant_type	是	获取 access_token 填写 client_credential
appid	是	第三方用户唯一凭证
secret	是	第三方用户唯一凭证密钥，即 appsecret

正常情况下，会返回下述 JSON 数据包给开发者：

```
{"access_token": "ACCESS_TOKEN", "expires_in": 7200}
```

access_token 是获取到的凭证，expires_in 是凭证有效时间，单位：s。

错误时微信会返回错误码等信息，JSON 数据包示例如下（该示例为 AppID 无效错误）：

```
{"errcode": 40013, "errmsg": "invalid appid"}
```

5.6.2.2 发送模板消息

接口地址：ACCESS_TOKEN 需换成上文获取到的 access_token，请求地址是：

```
https://api.weixin.qq.com/cgi-bin/message/wxopen/template/send?access_token=ACCESS_TOKEN
```

HTTP 请求方式：POST。

POST 参数说明如表 5-16 所示。

表 5-16　　POST 参数说明

参数	必填	说明
touser	是	接收者（用户）的 openid
template_id	是	所需下发的模板消息的 id
page	否	单击模板卡片后的跳转页面，仅限本小程序内的页面。支持带参数（示例 index?foo=bar）。该字段不填则模板无跳转
form_id	是	表单提交场景下，为 submit 事件带上的 formId；支付场景下，为本次支付的 prepay_id
data	是	模板内容，不填则下发空模板
color	否	模板内容字体的颜色，不填默认黑色
emphasis_keyword	否	模板需要放大的关键词，不填则默认无放大

示例代码如下所示：

```
{
  "touser": "OPENID",
  "template_id": "TEMPLATE_ID",
  "page": "index",
  "form_id": "FORMID",
  "data": {
      "keyword1": {
          "value": "339208499",
          "color": "#173177"
      },
      "keyword2": {
          "value": "2016 年 12 月 05 日 12:30",
          "color": "#173177"
      },
      "keyword3": {
          "value": "微信大厦",
```

```
            "color": "#173177"
        } ,
        "keyword4": {
            "value": "广州市海珠区海珠路 111",
            "color": "#173177"
        }
    },
    "emphasis_keyword": "keyword1.DATA"
}
```

在调用模板消息接口后，会返回 JSON 数据包。正常时返回 JSON 数据包示例：

```
{
  "errcode": 0,
  "errmsg": "ok",
}
```

错误时会返回错误码信息，说明如表 5-17 所示。

表 5-17　　错误码说明

错误码	说明
40037	template_id 不正确
41028	form_id 不正确，或者已过期
41029	form_id 已被使用
41030	page 不正确
45009	接口调用超过限额（目前默认每个账号日调用限额为 100 万元）

上面代码发送之后，用户收到的模板消息，显示的效果如图 5-8 所示。

- 下发条件说明

支付

当用户在小程序内完成支付行为，小程序允许开发者向用户在 7 天内推送有限条数的模板消息（1 次支付可下发 1 条，多次支付下发条数独立，互相不影响）。

提交表单

当用户在小程序内发生过提交表单行为且该表单声明为要发模板消息的，开发者需要向用户提供服务时，小程序允许开发者向用户在 7 天内推送有限条数的模板消息（1 次提交表单可下发 1 条，多次提交下发条数独立，相互不影响）。

- 审核说明

1. 标题

1.1　标题不能相同

1.2　标题意思不能存在过度相似

图 5-8　模板消息显示效果

1.3 标题必须以“提醒”或“通知”结尾

1.4 标题不能带特殊符号、个性化字词等没有行业通用性的内容

1.5 标题必须能体现具体服务场景

1.6 标题不能涉及营销相关内容，包括但不限于：

消费优惠类、购物返利类、商品更新类、优惠券类、代金券类、红包类、会员卡类、积分类、活动类等营销倾向通知

2. 关键词

2.1 同一标题下，关键词不能相同

2.2 同一标题下，关键词不能存在过度相似

2.3 关键词不能带特殊符号、个性化字词等没有行业通用性的内容

2.4 关键词内容示例必须与关键词对应匹配

2.5 关键词不能太过宽泛，需要具有限制性，例如：“内容”这个词就太宽泛，不能通过审核

- 违规说明

除不能违反运营规范外，还不能违反以下规则，包括但不限于：

不允许恶意诱导用户进行触发操作，以达到可向用户下发模板目的

不允许恶意骚扰，下发对用户造成骚扰的模板

不允许恶意营销，下发营销目的模板

不允许通过服务号下发模板来告知用户在小程序内触发的服务相关内容

- 处罚说明

根据违规情况给予相应梯度的处罚，一般处罚规则如下：

第一次违规，删除违规模板以示警告

第二次违规，封禁接口 7 天

第三次违规，封禁接口 30 天

第四次违规，永久封禁接口

处罚结果及原因以站内信形式告知。

5.7 客服消息

5.7.1 接收消息和事件

在页面组件中我们讲过 contact-button 组件（客服会话按钮），该组件用于在页面上显示一个客服会话按钮，用户单击该按钮后会进入客服会话。当用户在客服会话发送消息（或进

行某些特定的用户操作引发的事件推送）时，微信服务器将发送消息（或事件）的数据包（JSON 或者 XML 格式）POST 请求到开发者填写的 URL。开发者收到请求后可以使用发送客服消息接口进行异步回复。

微信服务器在将用户的消息发给小程序的开发者服务器地址（开发设置处配置）后，如果微信服务器在 5s 内收不到响应就会断掉连接，并且重新发起请求，总共重试 3 次。如果在调试中，开发者发现用户无法收到响应的消息，可以检查是否消息处理超时。关于重试的消息排重，有 MsgId 的消息推荐使用 MsgId 排重。事件类型消息推荐使用 FromUserName + CreateTime 排重。

服务器收到请求必须做出下述回复，这样微信服务器才不会对此做任何处理，并且不会发起重试，否则，将出现严重的错误提示。

1. 直接回复 success（推荐方式）。
2. 直接回复空串（指字节长度为 0 的空字符串，而不是结构体中 content 字段的内容为空）。

一旦遇到以下情况，微信都会在小程序会话中向用户下发系统提示“该小程序客服暂时无法提供服务，请稍后再试”：

1. 开发者在 5s 内未回复任何内容。
2. 开发者回复了异常数据。

如果希望增强安全性，可以在开发者中心处开启消息加密，这样，用户发给小程序的消息以及小程序被动回复用户消息都会继续加密，详见微信开放平台 — 资源中心 — 第三方平台 —消息加解密说明 — 消息加解密接入指引。

各消息类型的推送 JSON、XML 数据包结构如下。

文本消息，用户在客服会话中发送文本消息时将产生如下数据包。

XML 格式：

```
<xml>
   <ToUserName><![CDATA[toUser]]></ToUserName>
   <FromUserName><![CDATA[fromUser]]></FromUserName>
   <CreateTime>1482048670</CreateTime>
   <MsgType><![CDATA[text]]></MsgType>
   <Content><![CDATA[this is a test]]></Content>
   <MsgId>1234567890123456</MsgId>
</xml>
```

JSON 格式：

```
{
    "ToUserName": "toUser",
    "FromUserName": "fromUser",
    "CreateTime": 1482048670,
    "MsgType": "text",
    "Content": "this is a test",
    "MsgId": 1234567890123456
}
```

文本消息参数说明如表 5-18 所示。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 5-18　　文本消息参数说明

参数	说明
ToUserName	小程序的原始 id
FromUserName	发送者的 openid
CreateTime	消息创建时间（整型）
MsgType	text
Content	文本消息内容
MsgId	消息 id，64 位整型

图片消息，用户在客服会话中发送图片消息时将产生如下数据包。

XML 格式：

```
<xml>
     <ToUserName><![CDATA[toUser]]></ToUserName>
     <FromUserName><![CDATA[fromUser]]></FromUserName>
     <CreateTime>1482048670</CreateTime>
     <MsgType><![CDATA[image]]></MsgType>
     <PicUrl><![CDATA[this is a url]]></PicUrl>
     <MediaId><![CDATA[media_id]]></MediaId>
     <MsgId>1234567890123456</MsgId>
</xml>
```

JSON 格式：

```
{
    "ToUserName": "toUser",
    "FromUserName": "fromUser",
    "CreateTime": 1482048670,
    "MsgType": "image",
    "PicUrl": "this is a url",
    "MediaId": "media_id",
    "MsgId": 1234567890123456
}
```

图片消息参数说明如表 5-19 所示。

表 5-19　　图片消息参数说明

参数	说明
ToUserName	小程序的原始 id
FromUserName	发送者的 openid
CreateTime	消息创建时间（整型）
MsgType	image
PicUrl	图片链接（由系统生成）
MediaId	图片消息媒体 id，可以调用获取临时素材接口拉取数据
MsgId	消息 id，64 位整型

进入会话事件，用户在单击小程序“客服会话按钮”进入客服会话时将产生如下数据包。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

XML 格式：

```
<xml>
    <ToUserName><![CDATA[toUser]]></ToUserName>
    <FromUserName><![CDATA[fromUser]]></FromUserName>
    <CreateTime>1482048670</CreateTime>
    <MsgType><![CDATA[event]]></MsgType>
    <Event><![CDATA[user_enter_tempsession]]></Event>
    <SessionFrom><![CDATA[sessionFrom]]></SessionFrom>
</xml>
```

JSON 格式：

```
{
    "ToUserName": "toUser",
    "FromUserName": "fromUser",
    "CreateTime": 1482048670,
    "MsgType": "event",
    "Event": "user_enter_tempsession",
    "SessionFrom": "sessionFrom"
}
```

进入会话事件消息参数说明如表 5-20 所示。

表 5-20　　进入会话事件消息参数说明

参数	说明
ToUserName	小程序的原始 id
FromUserName	发送者的 openid
CreateTime	事件创建时间（整型）
MsgType	event
Event	事件类型，user_enter_tempsession
SessionFrom	开发者在客服会话按钮设置的 sessionFrom 参数

5.7.2 发送客服消息

当用户和小程序客服产生特定动作的交互时（具体动作列表请见表 5-21），微信将会把消息数据推送给开发者，开发者可以在一段时间内（目前修改为 48 小时）调用客服接口，通过 POST 一个 JSON 数据包来发送消息给普通用户。此接口主要用于客服等有人工消息处理环节的功能，方便开发者为用户提供更加优质的服务。

不同动作触发后，客服接口允许的下发消息条数和下发时限不同。下发条数达到上限后，会收到错误返回码，目前允许的动作列表如表 5-21 所示。

表 5-21　　动作列表

用户动作	允许下发条数限制	下发时限
用户通过客服消息按钮进入会话	1 条	1 分钟
用户发送信息	3 条	48 小时

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

客服接口发消息使用 HTTP 请求方式：POST，接口地址：

```
https://api.weixin.qq.com/cgi-bin/message/custom/send?access_token=ACCESS_TOKEN
```

各消息类型所需的 JSON 数据包如下：

发送文本消息：

```
{
    "touser":"OPENID",
    "msgtype":"text",
    "text":
    {
         "content":"Hello World"
    }
}
```

发送图片消息：

```
{
    "touser":"OPENID",
    "msgtype":"image",
    "image":
    {
      "media_id":"MEDIA_ID"
    }
}
```

参数说明如表 5-22 所示。

表 5-22　　参数说明

参数	是否必须	说明
access_token	是	调用接口凭证
touser	是	普通用户
Msgtype	是	消息类型，文本为 text，图片为 image
content	是	文本消息内容
media_id	是	发送的图片的媒体 id，通过新增素材接口上传图片文件获得

返回码说明如表 5-23 所示。

表 5-23　　返回码说明

返回码	说明
-1	系统繁忙，此时请开发者稍候再试
0	请求成功
40001	获取 access_token 时 AppSecret 错误，或者 access_token 无效。请开发者认真比对 AppSecret 的正确性，或查看是否正在为恰当的小程序调用接口
40002	不合法的凭证类型
40003	不合法的 openid，请开发者确认 openid 是否是其他小程序的 openid
45015	回复时间超过限制
45047	客服接口下行条数超过上限
48001	api 功能未授权，请确认小程序已获得该接口

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

5.7.3 临时素材接口

1）获取临时素材

小程序可以使用本接口获取客服消息内的临时素材（即下载临时的多媒体文件）。目前小程序仅支持下载图片文件。

HTTP 请求方式：GET，HTTPS 调用具体地址如下所示：

```
    https://api.weixin.qq.com/cgi-bin/media/get?access_token=ACCESS_
TOKEN&media_id=MEDIA_ID
```

参数说明如表 5-24 所示。

表 5-24　　参数说明

参数	是否必须	说明
access_token	是	调用接口凭证
media_id	是	媒体文件 id

正确情况下的返回 HTTP 如下所示：

```
    HTTP/1.1 200 OK
    Connection: close
    Content-Type: image/jpeg
    Content-disposition: attachment; filename="MEDIA_ID.jpg"
    Date: Sun, 06 Jan 2013 10:20:18 GMT
    Cache-Control: no-cache, must-revalidate
    Content-Length: 339721
    curl  -G  "https://api.weixin.qq.com/cgi-bin/media/get?access_token=ACCESS_
TOKEN&media_id=MEDIA_ID"
```

如果返回的是视频消息素材，则内容如下所示：

```
    {
     "video_url":DOWN_URL
    }
```

错误情况下的返回 JSON 数据包示例如下所示（示例为无效媒体 ID 错误）：

```
    {
      "errcode":40007,
      "errmsg":"invalid media_id"
    }
```

2）新增临时素材

小程序可以使用本接口把媒体文件（目前仅支持图片）上传到微信服务器，用户发送客服消息或被动回复用户消息。

HTTP 请求方式：POST/FORM，HTTPS 调用接口如下所示：

```
    https://api.weixin.qq.com/cgi-bin/media/upload?access_token=ACCESS_TOKEN&type=TYPE
```

参数说明如表 5-25 所示。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 5-25 参数说明

参数	是否必须	说明
access_token	是	调用接口凭证
type	是	image
media	是	form-data 中媒体文件标识，有 filename、filelength、content-type 等信息

正确情况下的返回 JSON 数据包结果如下所示：

```
{
  "type":"TYPE",
  "media_id":"MEDIA_ID",
  "created_at":123456789
}
```

参数说明如表 5-26 所示。

表 5-26 参数说明

参数	说明
type	image
media_id	媒体文件上传后，获取标识
created_at	媒体文件上传时间戳

错误情况下的返回 JSON 数据包示例如下所示（示例为无效媒体类型错误）：

```
{
  "errcode":40004,
  "errmsg":"invalid media type"
}
```

5.7.4 接入指引

接入微信小程序消息服务，开发者需要按照如下步骤完成：

1. 填写服务器配置。
2. 验证服务器地址的有效性。
3. 依据接口文档实现业务逻辑。

下面详细介绍这 3 个步骤。

第 1 步：填写服务器配置

登录微信小程序官网后，单击客服消息来添加客服人员，可以添加 100 个。如图 5-9 所示。

找到小程序官网的“设置-开发设置-消息推送”页面。启用并设置消息推送配置后，用户发给小程序的消息以及开发者需要的事件推送，都将被微信转发至该服务器地址中。管理员扫码启用消息服务，填写服务器地址（URL）、Token 和 EncodingAESKey。URL 是开发者用

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

来接收微信消息和事件的接口 URL；Token 可由开发者任意填写，用作生成签名（该 Token 会和接口 URL 中包含的 Token 进行比对，从而验证安全性）；EncodingAESKey 由开发者手动填写或随机生成，将用作消息体加解密密钥。如图 5-10 所示。

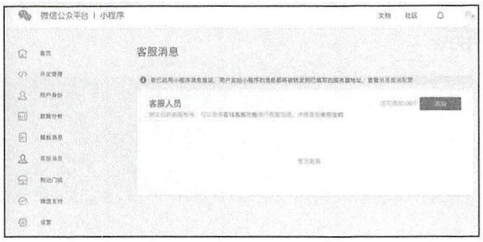


图 5-9 添加客服消息

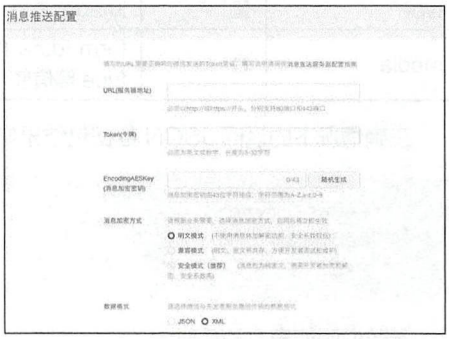


图 5-10 服务器配置

开发者可选择消息加解密方式：明文模式、兼容模式和安全模式，还可以选择消息数据格式：XML 格式或 JSON 格式。加密方式的默认状态是明文格式，而数据格式的默认状态是 XML 格式。模式的选择与服务器配置在提交后都会立即生效，请开发者谨慎填写及选择。切换加密方式和数据格式需要提前配置好相关代码。

第 2 步：验证服务器地址的有效性

开发者提交信息后，微信服务器将发送 GET 请求到填写的服务器地址 URL 上，GET 请求携带参数如表 5-27 所示。

表 5-27 GET 请求的参数说明

参数	说明
signature	微信加密签名，signature 结合了开发者填写的 token 参数和请求中的 timestamp 参数、nonce 参数
timestamp	时间戳
nonce	随机数
echostr	随机字符串

开发者通过检验 signature 对请求进行校验（下面有校验方式）。若确认此次 GET 请求来自微信服务器，请原样返回 echostr 参数内容，则接入生效，成为开发者成功，否则接入失败。

C#服务器直接返回 echostr 参数内容，代码返回如下所示：

```
public partial class Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
            string token = "";

            try
            {
                token = Request.QueryString["echostr"].ToString();
            }
            catch { }

            Response.Write(token);
        }
    }
```

加密/校验流程如下：

1. 将 token、timestamp、nonce 3 个参数进行字典序排序。
2. 将 3 个参数字符串拼接成一个字符串进行 sha1 加密。
3. 开发者获得加密后的字符串可与 signature 对比，标识该请求来源于微信检验。

signature 的 PHP 示例代码如下所示：

```
private function checkSignature()
{
    $signature = $_GET["signature"];
    $timestamp = $_GET["timestamp"];
    $nonce = $_GET["nonce"];

    $token = TOKEN;
    $tmpArr = array($token, $timestamp, $nonce);
    sort($tmpArr, SORT_STRING);
    $tmpStr = implode( $tmpArr );
    $tmpStr = sha1( $tmpStr );

    if( $tmpStr == $signature ){
        return true;
    }else{
        return false;
    }
}
```

PHP 示例代码下载地址如下所示：

```
https://wximg.gtimg.com/shake_tv/mpwiki/cryptoDemo.zip
```

第 3 步：依据接口文档实现业务逻辑

验证 URL 有效性成功后即接入生效，成为开发者。至此，用户向小程序客服发送消息，或者进入会话等情况时，开发者填写的服务器配置 URL 将得到微信服务器推送过来的消息和事件，开发者可以依据自身业务逻辑进行响应。开发者所填写的 URL 必须以 http:// 或 https:// 开头，分别支持 80 端口和 443 端口。

5.7.5 转发消息

如果小程序设置了消息推送，普通微信用户向小程序客服发消息时，微信服务器会先将消

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

息 POST 到开发者填写的 URL 上，如果希望将消息转发到网页版客服工具，则需要开发者在响应包中返回 MsgType 为 transfer_customer_service 的消息，微信服务器收到响应后会把当次发送的消息转发至客服系统。

用户被客服接入以后，客服关闭会话以前，处于会话过程中时，用户发送的消息均会被直接转发至客服系统。当会话超过 30 分钟客服没有关闭时，微信服务器会自动停止转发至客服，而将消息恢复发送至开发者填写的 URL 上。

用户在等待队列中时，用户发送的消息仍然会被推送至开发者填写的 URL 上。

这里特别要注意，微信服务器只针对微信用户发来的消息才进行转发，而对于其他事件（比如用户从小程序唤起客服会话）都不应该转发，否则客服在客服系统上就会看到一些无意义的消息了。

消息转发到网页版客服工具

开发者只在响应包中返回 MsgType 为 transfer_customer_service 的消息，微信服务器收到响应后就会把当次发送的消息转发至客服系统。

示例代码如下所示：

```
    <xml>
    <ToUserName><![CDATA[touser]]></ToUserName>    <FromUserName><![CDATA[fromuser]]>
</FromUserName>      <CreateTime>1399197672</CreateTime>      <MsgType><![CDATA
[transfer_customer_service]]></MsgType>
    </xml>
```

参数说明如表 5-28 所示。

表 5-28　　参数说明

参数	是否必须	说明
ToUserName	是	接收方账号（收到的 openid）
FromUserName	是	开发者微信号
CreateTime	是	消息创建时间（整型）
MsgType	是	transfer_customer_service

5.7.6　客服输入状态

开发者可通过调用“客服输入状态接口”，返回客服当前输入状态给用户。

1. 此接口需要客服消息接口权限。
2. 如果不满足发送客服消息的触发条件，则无法下发输入状态。
3. 下发输入状态，需要客服在之前 30s 内跟用户有过消息交互。
4. 在输入状态中（持续 15s），不可重复下发输入态。
5. 在输入状态中，如果向用户下发消息，会同时取消输入状态。

HTTP 请求方式：POST，接口调用请求说明：

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
https://api.weixin.qq.com/cgi-bin/message/custom/typing? access_token=
ACCESS_TOKEN
```

JSON 数据包如下所示：

```
{
"touser":"OPENID",
"command":"Typing"
}
```

预期返回如下所示：

```
{
"errcode":0,
"errmsg":"ok"
}
```

参数说明如表 5-29 所示。

表 5-29　　参数说明

参数	是否必须	说明
access_token	是	调用接口凭证
touser	是	普通用户（openid）
command	是	"Typing"：对用户下发“正在输入”状态；"CancelTyping"：取消对用户的“正在输入”状态

返回码说明如表 5-30 所示。

表 5-30　　返回码说明

返回码	说明
45072	command 字段取值不对
45080	下发输入状态，需要在之前 30s 内跟用户有过消息交互
45081	已经在输入状态，不可重复下发

5.8　转发（分享）

1）onShareAppMessage（options）在 Page 中定义 onShareAppMessage 函数，设置该页面的分享信息。

- 只有定义了此事件处理函数，右上角菜单才会显示“分享”按钮。
- 用户单击分享按钮的时候会调用。
- 此事件需要 return 一个 Object，用于自定义分享内容。

options 参数说明如表 5-31 所示。

非卖品，仅供非商业用途或交流学习使用

表 5-31 options 参数说明

参数	类型	说明	最低版本
from	String	转发事件来源。button：页面内转发按钮；menu：右上角转发菜单	1.2.4
target	Object	如果 from 的值是 button，则 target 是触发这次转发事件的 button，否则为 undefined	1.2.4

自定义分享字段如表 5-32 所示。

表 5-32 自定义字段

字段	说明	默认值
title	分享标题	当前小程序名称
path	分享路径	当前页面 path，必须是以 / 开头的完整路径
success	分享成功的回调函数	
fail	分享失败的回调函数	
complete	分享结束的回调函数（分享成功、失败都会执行）	

回调结果如表 5-33 所示。

表 5-33 回调结果

回调类型	errMsg	说明
success	shareAppMessage:ok	分享成功
fail	shareAppMessage:fail cancel	用户取消分享
detail	shareAppMessage:fail (detail message)	分享失败，其中 detail message 为详细失败信息

success 回调参数说明如表 5-34 所示。

表 5-34 success 回调参数说明

参数	类型	说明	最低版本
shareTickets	StringArray	shareTicket 数组，每一项是一个 shareTicket，对应一个转发对象	1.1.0

使用代码如下所示：

```
//index.js
Page({
  onShareAppMessage: function (res) {
    if (res.from === 'button') {
      // 来自页面内转发按钮
```

非卖品，仅供非商业用途或交流学习使用

```
      console.log(res.target)
    }
    return {
      title: '自定义分享标题',
      path: '/pages/index/index?id=123',
      success: function(res) {
        // 分享成功
      },
      fail: function(res) {
        // 分享失败
      }
    }
  },
  data: {
  },
  onLoad: function () {
    console.log('onLoad')
  }
})
```

2）wx.showShareMenu（OBJECT）：显示分享按钮。从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持，低版本需做兼容处理。showShareMenu 的 OBJECT 参数说明如表 5-35 所示。

表 5-35　　showShareMenu 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码如下所示：

```
//显示分享按钮
  testButtonClick1: function () {
    wx.showShareMenu({
      success: function(res) {
        // 显示成功
      },
      fail: function(res) {
        // 显示失败
      },
      complete: function(res) {
        // 显示完成
      },
    })
  },
```

3）wx.hideShareMenu（OBJECT）：隐藏分享按钮。从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持，低版本需做兼容处理。hideShareMenu 的 OBJECT 参数说明如表 5-36 所示。

非卖品，仅供非商业用途或交流学习使用

表 5-36 hideShareMenu 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用代码如下所示：

```
//隐藏分享按钮
  testButtonClick1: function () {
    wx.hideShareMenu({
      success: function(res) {
        // 隐藏成功
      },
      fail: function(res) {
        // 隐藏失败
      },
      complete: function(res) {
        // 隐藏完成
      },
    })
  },
```

分享图片不能自定义；小程序会取当前页面，从顶部开始，高度为 80%屏幕宽度的图像作为分享图片。Page.onShareAppMessage 的回调函数从微信客户端 6.5.7 版本才开始支持。

4）wx.updateShareMenu(OBJECT)：更新转发属性，从基础库 1.2.0 开始支持。updateShareMenu 的 OBJECT 参数说明如表 5-37 所示。

表 5-37 updateShareMenu 的 OBJECT 参数说明

参数	类型	必填	说明
withShareTicket	Boolean	否	是否使用带 shareTicket 的转发详情
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码如下所示：

```
wx.updateShareMenu({
    withShareTicket: true,
    success() {
    }
})
```

5）wx.getShareInfo(OBJECT)：获取转发详细信息，从基础库 1.1.0 开始支持。getShareInfo 的 OBJECT 参数说明如表 5-38 所示。

非卖品，仅供非商业用途或交流学习使用

表 5-38 getShareInfo 的 OBJECT 参数说明

参数	类型	必填	说明
shareTicket	String	是	shareTicket
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

CALLBACK 参数说明如表 5-39 所示。

表 5-39 CALLBACK 参数说明

参数	类型	说明
errMsg	String	错误信息
encryptedData	String	包括敏感数据在内的完整转发信息的加密数据，详见加密数据解密算法
iv	String	加密算法的初始向量，详见加密数据解密算法

encryptedData 解密后为一个 JSON 结构，包含字段如表 5-40 所示。

表 5-40 encryptedData 包含字段

字段	说明
openGId	群对当前小程序的唯一 ID

6）获取更多转发信息

通常我们希望转发出去的小程序被二次打开的时候能够获取到一些信息，例如群的标识。现在我们通过调用 wx.showShareMenu 并且设置 withShareTicket 为 true，当用户将小程序转发到任一群聊之后，可以获取到此次转发的 shareTicket，此转发卡片在群聊中被其他用户打开时，可以在 App.onLaunch() 或 App.onShow() 获取到另一个 shareTicket。这两步获取到的 shareTicket 均可通过 wx.getShareInfo 接口获取到相同的转发信息。

7）页面内发起转发

从基础库版本 1.2.0 开始支持，开发者通过给 button 组件设置属性 open-type="share"，可以在用户单击按钮后触发 Page.onShareAppMessage() 事件，如果当前页面没有定义此事件，则单击后无效果。相关组件：button。

8）使用指引

转发按钮，旨在帮助用户更流畅地与好友分享内容和服务。转发应是用户自发的行为，且在需要时触手可及。开发者在使用时若遵从以下指引，会得到更佳的用户体验。

含义清晰：明确、一目了然的图形按钮，将为用户减少理解的时间。在微信的资源下载中心，我

非卖品，仅供非商业用途或交流学习使用

们可以找到这样的按钮素材并直接使用。或者我们可以根据自己业务的设计风格，灵活设计含义清晰的按钮的样式。当然，我们也可以直接使用带文字的按钮，“转发给好友”也足够明确。

方便单击：按钮单击热区不宜过小，亦不宜过大。同时，转发按钮与其他按钮一样，热区也不宜过密，以免用户误操作。

按需出现：并非所有页面都适合放置转发按钮，涉及用户隐私的非公开内容，或可能打断用户完成当前操作体验的场景，该功能并不推荐使用。同时，由于转发过程中微信将截取用户屏幕图像作为配图，因此，需要注意帮助用户屏蔽个人信息。

尊重意愿：理所当然，并非所有的用户都喜欢与朋友分享你的小程序。因此，转发不应该成为一个诱导或强制行为，如转发后才能解锁某项功能等。请注意，这类做法不仅不被推荐，还可能违反微信的《运营规范》，微信强烈建议你在使用前阅读这部分内容。

以上内容陈列了最重要的几点，如果你有时间，可以完整浏览《设计指南》，相信会有更多的收获。

注意事项：

1. 不自定义转发图片的情况下，小程序默认取当前页面，从顶部开始，高度为 80% 屏幕宽度的图像作为转发图片。

2. 转发的调试支持请查看普通转发的调试支持和带 shareTicket 的转发支持。

3. 只有转发到群聊中打开才可以获取到 shareTickets 返回值，单聊没有 shareTickets。

4. shareTicket 仅在当前小程序生命周期内有效。

5. 由于策略变动，小程序群相关功能需进行调整，开发者可先使用 wx.getShareInfo 接口中的群 ID 进行功能开发。

5.9 二维码

开发者通过后台接口可以获取小程序任意页面的二维码，扫描该二维码可以直接进入小程序对应的页面。

接口地址：

```
    https://api.weixin.qq.com/cgi-bin/wxaapp/createwxaqrcode?access_
token=ACCESS_TOKEN
```

POST 参数说明如表 5-41 所示。

表 5-41　　POST 参数说明

参数	默认值	说明
path		不能为空，最大长度 128 字节
width	430	二维码的宽度

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

开发者可以使用 AppID 和 AppSecret 调用本接口来获取 access_token。AppID 和 AppSecret 可登录微信公众平台官网-设置-开发设置中获得。AppSecret 生成后请自行保存，因为在公众平台每次生成查看都会导致 AppSecret 被重置。注意调用所有微信接口时均需使用 https 协议。如果第三方不使用中控服务器，而是选择各个业务逻辑点各自去刷新 access_token，那么就可能会产生冲突，导致服务不稳定。

请求接口地址：

```
https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=APPID&secret=APPSECRET
```

示例代码如下所示：

```
{"path": "pages/index?query=1", "width": 430}
```

通过该接口仅能生成已发布的小程序的二维码，开发者可以在开发者工具预览时生成开发版的带参数二维码。带参数二维码只有 100 000 个，请谨慎调用。POST 参数需要转成 json 字符串，不支持 form 表单提交。

5.10 收货地址

wx.chooseAddress（OBJECT）：调起用户编辑收货地址原生界面，并在编辑完成后返回用户选择的地址。从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持，低版本需做兼容处理。chooseAddress OBJECT 参数说明如表 5-42 所示。

表 5-42 chooseAddress 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	返回用户选择的收货地址信息
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 5-43 所示。

表 5-43 success 返回参数说明

参数	类型	说明
errMsg	String	调用结果
userName	String	收货人姓名
postalCode	String	邮编
provinceName	String	国标收货地址第一级地址
cityName	String	国标收货地址第二级地址

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

续表

参数	类型	说明
CountyName	String	国标收货地址第三级地址
detailInfo	String	详细收货地址信息
nationalCode	String	收货地址国家码
telNumber	String	收货人手机号码

使用代码如下所示：

```
<!--index.wxml-->
<button bindtap="testButtonClick" class="section">收货地址</button>

//index.js
Page({
  data: {
  },

  //收货地址
  testButtonClick: function () {

    wx.chooseAddress({
      success: function (res) {
        console.log(res.userName)
        console.log(res.postalCode)
        console.log(res.provinceName)
        console.log(res.cityName)
        console.log(res.countyName)
        console.log(res.detailInfo)
        console.log(res.nationalCode)
        console.log(res.telNumber)
      }
    })

  },
  onLoad: function () {
    console.log('onLoad')
  }
})
```

5.11 卡券

1）wx.addCard（OBJECT）：批量添加卡券。从基础库版本 1.1.0 和微信客户端 6.5.4 版本开始支持，低版本需做兼容处理。目前开发者工具上尚未支持小程序卡券的调试，请在真机上调试。addCard 的 OBJECT 参数说明如表 5-44 所示。

表 5-44　　addCard 的 OBJECT 参数说明

参数	类型	必填	说明
cardList	ObjectArray	是	需要添加的卡券列表
success	Function	否	接口调用成功的回调函数

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

续表

参数	类型	必填	说明
Fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

回调结果说明如表 5-45 所示。

表 5-45 回调结果说明

回调类型	errMsg	说明
success	addCard:ok	添加卡券成功
fail	addCard:fail cancel	用户取消添加卡券
fail	addCard:fail (detail message)	添加卡券失败，其中 detail message 为后台返回的详细失败原因

使用代码如下所示：

```
    //批量添加卡券
      testButtonClick1: function () {

        wx.addCard({
          cardList: [
            {
              cardId: '1',
              cardExt:  '{"code":  "123",  "openid":  "1108",  "timestamp":
"1230048689", "signature":"6789"}'
            }, {
              cardId: '2',
              cardExt:  '{"code":  "234",  "openid":  "1109",  "timestamp":
"1230048889", "signature":"56789"}'
            }
          ],
          success: function (res) {
            console.log(res.cardList) // 卡券添加结果
          },
          fail: function (res) {
            console.log(res) // 卡券添加结果
          }
        })
      },
```

2）wx.openCard（OBJECT）：查看微信卡包中的卡券。从基础库版本 1.1.0 和微信客户端 6.5.4 版本开始支持，低版本需做兼容处理。openCard 的 OBJECT 参数说明如表 5-46 所示。

表 5-46 openCard 的 OBJECT 参数说明

参数	类型	必填	说明
cardList	ObjectArray	是	需要打开的卡券列表
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

使用代码如下所示：

```
// 查看微信卡包中的卡券
  testButtonClick2: function () {

    wx.openCard({
      cardList: [
        {
          cardId: '',
          code: ''
        }, {
          cardId: '',
          code: ''
        }
      ],
      success: function (res) {
      }
    })
  },
```

5.12 设置

1）wx.openSetting（OBJECT）：调起客户端小程序设置界面，返回用户设置的操作结果。从基础库版本 1.1.0 和微信客户端 6.5.6 版本开始支持，低版本需做兼容处理。openSetting 的 OBJECT 参数说明如表 5-47 所示。

表 5-47 openSetting 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数，返回内容详见返回参数说明
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 5-48 所示。

表 5-48 success 返回参数说明

参数	类型	说明
authSetting	Object	用户授权结果，其中 key 为 scope 值，value 为 Bool 值，表示用户是否允许授权

scope 参数说明如表 5-49 所示。

表 5-49 scope 参数说明

scope	对应接口
scope.userInfo	wx.getUserInfo
scope.userLocation	wx.getLocation、wx.chooseLocation
scope.address	wx.chooseAddress
scope.record	wx.startRecord

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

使用代码如下所示：

```
//设置
  testButtonClick: function () {

    wx.openSetting({
      success: (res) => {
        console.log('authSetting = ' + res.authSetting["scope.userInfo"])
        /*
         * res.authSetting = {
         *   "scope.userInfo": true,
         *   "scope.userLocation": true
         * }
         */
      },
      fail: function (res) {

      },
      complete: function (res) {

      }
    })
  },
```

2）wx.getSetting(OBJECT)：获取用户的当前设置，从基础库版本 1.2.0 开始支持。getSetting 的 OBJECT 参数说明如表 5-50 所示。

表 5-50　　getSetting 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数，返回内容详见返回参数说明
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 5-51 所示。

表 5-51　　success 返回参数说明

参数	类型	说明
authSetting	Object	用户授权结果，其中 key 为 scope 值，value 为 Bool 值，表示用户是否允许授权，详见 scope 参数说明（表 5-49）

示例代码如下所示：

```
wx.getSetting({
  success: (res) => {
  /*
   * res.authSetting = {
  *   "scope.userInfo": true,
  *   "scope.userLocation": true
  * }
  */
 }
})
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

5.13 微信运动

基础库版本 1.2.0 开始支持 wx.getWeRunData(OBJECT)，获取用户过去 30 天的微信运动步数，需要先调用 wx.login 接口。

getWeRunData 的 OBJECT 参数说明如表 5-52 所示。

表 5-52 getWeRunData 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 5-53 所示。

表 5-53 success 返回参数说明

参数	类型	说明
errMsg	String	调用结果
encryptedData	String	包括敏感数据在内的完整用户信息的加密数据，详见加密数据解密算法
iv	String	加密算法的初始向量，详见加密数据解密算法

示例代码如下所示：

```
wx.getWeRunData({
  success(res) {
    const encryptedData = res.encryptedData
  }
})
```

encryptedData 解密后为以下 json 结构，包含字段如表 5-54 所示。

表 5-54 encryptedData 结构

属性	类型	说明
stepInfoList	ObjectArray	用户过去 30 天的微信运动步数

stepInfoList 结构如表 5-55 所示。

表 5-55 stepInfoList 结构

属性	类型	说明
timestamp	Number	时间戳，表示数据对应的时间
step	Number	微信运动步数

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

输出显示如下所示：

```
{
  "stepInfoList": [
    {
      "timestamp": 1445866601,
      "step": 100
    },
    {
      "timestamp": 1445866602,
      "step": 100
    }
  ]
}
```

5.14 打开小程序

1）wx.navigateToMiniProgram(OBJECT)

基础库版本 1.3.0 开始支持 wx.navigateToMiniProgram(OBJECT)，打开同一公众号下关联的另一个小程序。iOS 客户端 6.5.9 版本开始支持，Android 客户端即将在 6.5.10 版本开始支持，请先使用 iOS 客户端进行调试。navigateToMiniProgram 的 OBJECT 参数说明如表 5-56 所示。

表 5-56 navigateToMiniProgram 的 OBJECT 参数说明

参数	类型	必填	说明
appId	String	是	要打开的小程序 appId
path	String	否	打开的页面路径，如果为空则打开首页
extradata	Object	否	需要传递给目标小程序的数据，目标小程序可在 App.onLaunch()、App.onShow() 中获取到这份数据
envVersion	String	否	要打开的小程序版本，有效值 develop（开发版）、trial（体验版）和 release（正式版），仅在当前小程序为开发版时此参数有效；如果当前小程序是体验版或正式版，则打开的小程序必定是正式版。默认值 release
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 5-57 所示。

表 5-57 success 返回参数说明

参数	类型	说明
errMsg	String	调用结果

示例代码如下所示：

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
wx.navigateToMiniProgram({
  appId: '',
  path: 'pages/index/index?id=123',
  extraData: {
    foo: 'bar'
  },
  envVersion: 'develop',
  success(res) {
    // 打开成功
  }
})
```

2）wx.navigateBackMiniProgram(OBJECT)

基础库版本 1.3.0 开始支持 wx.navigateBackMiniProgram(OBJECT)，返回到上一个小程序，只有在当前小程序被其他小程序打开时可以调用成功。iOS 客户端 6.5.9 版本开始支持，Android 客户端即将在 6.5.10 版本开始支持，请先使用 iOS 客户端进行调试。

navigateBackMiniProgram 的 OBJECT 参数说明如表 5-58 所示。

表 5-58　　navigateBackMiniProgram 的 OBJECT 参数说明

参数	类型	必填	说明
extraData	Object	否	需要返回给上一个小程序的数据，上一个小程序可在 App.onShow() 中获取到这份数据
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 5-59 所示。

表 5-59　　success 返回参数说明

参数	类型	说明
errMsg	String	调用结果

示例代码如下所示：

```
wx.navigateBackMiniProgram({
  extraData: {
    foo: 'bar'
  },
  success(res) {
    // 返回成功
  }
})
```

5.15　打开 App

launchApp(OBJECT)打开 App。因为需要用户主动触发才能打开 App，所以该功能不由 API 来调用，而需要用 open-type 的值设置为 launchApp 的 <button> 组件的单击来触发。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

当小程序从 App 分享消息卡片的场景打开时（场景值 1036），小程序会获得打开 App 的能力，此时用户单击按钮可以打开分享该卡片的 App。即小程序不能打开任意 App，只能跳回分享该小程序卡片的 App。

在一个小程序的生命周期内，只有在特定条件下才具有打开 App 的能力。打开 App 的能力可以理解为小程序框架在内部管理的一个状态，为 true 则可以打开 App，为 false 则不可以打开 App。

在小程序的生命周期内，这个状态的初始值为 false，之后会随着小程序每次打开（无论是启动还是切换到前台）而改变：

- 当小程序从 1036（App 分享消息卡片）的场景打开时，该状态值为 true。
- 当小程序从 1089（微信聊天主界面下拉）或 1090（长按小程序右上角菜单唤出最近使用历史）的场景打开时，该状态不变，即保持上一次打开小程序时该状态的值。
- 当小程序从非 1036/1089/1090 的场景打开时，该状态值为 false。

小程序打开 App 的流程，如图 5-11 所示。

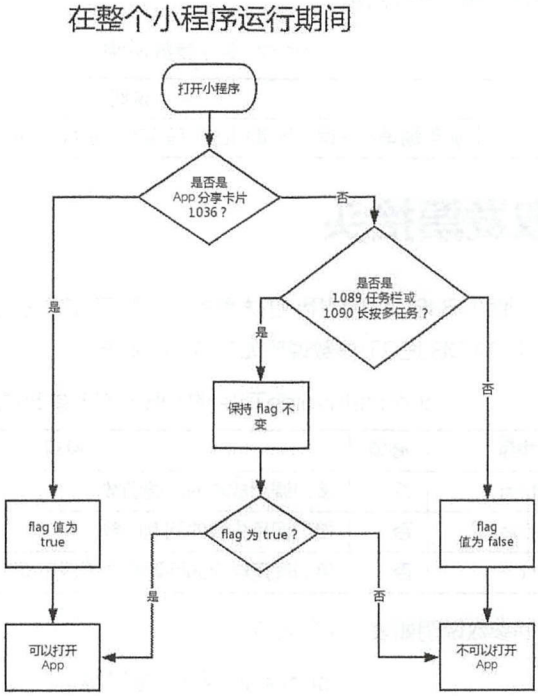


图 5-11 小程序打开 App 的流程

使用方法

1）小程序端

需要将 <button> 组件 open-type 的值设置为 launchApp。如果需要在打开 App 时向

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

App 传递参数，可以设置 app-parameter 为要传递的参数。开发者通过 binderror 可以监听打开 App 的错误事件。

2）app 端

Android 第三方 App 需要处理 ShowMessageFromWX.req 的微信回调，iOS 则需要将 appId 添加到第三方 App 工程所属的 plist 文件 URL types 字段。

示例代码如下所示：

```
    <!--index.wxml-->
    <button  open-type="launchApp"  app-parameter="wechat"  binderror="launch
AppError">打开 App</button>

    //index.js
    Page({
      launchAppError: function (e) {
        console.log("launchAppError:"+e.detail.errMsg)
      }
    })
```

error 事件参数说明如表 5-60 所示。

表 5-60　　error 事件参数说明

值	说明
invalid scene	调用场景不正确，即此时的小程序不具备打开 App 的能力

5.16　获取发票抬头

wx.chooseInvoiceTitle(OBJECT)选择用户的发票抬头，需要用户授权 scope. invoiceTitle。

chooseInvoiceTitle 的 OBJECT 参数说明如表 5-61 所示。

表 5-61　　chooseInvoiceTitle 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

其中 success 返回参数说明如表 5-62 所示。

表 5-62　　success 返回参数说明

参数	类型	说明
type	String	抬头类型（0：单位；1：个人）
title	String	抬头名称
TaxNumber	String	抬头税号

续表

参数	类型	说明
CompanyAddress	String	单位地址
telephone	String	手机号码
bankName	String	银行名称
bankAccount	String	银行账号
errMsg	String	接口调用结果

使用代码如下所示：

```
wx.chooseInvoiceTitle({
    success(res) {

      console.log("chooseInvoiceTitle:" + res.type)
      console.log("chooseInvoiceTitle:" + res.title)
      console.log("chooseInvoiceTitle:" + res)
    },
    fail(res) {

    },
    complete(res) {

    }
  })
```

5.17 生物认证

wx.checkIsSupportSoterAuthentication(OBJECT)用来获取本机支持的 SOTER 生物认证方式，从基础库版本 1.5.0 开始支持。

checkIsSupportSoterAuthentication 的 OBJECT 参数说明如表 5-63 所示。

表 5-63 checkIsSupportSoterAuthentication 的 OBJECT 参数说明

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

success 返回参数说明如表 5-64 所示。

表 5-64 success 返回参数说明

参数	类型	说明
supportMode	StringArray	该设备支持的、可被 SOTER 识别的生物识别方式
errMsg	String	接口调用结果

supportMode 的有效值如表 5-65 所示。

表 5-65　　supportMode 的有效值

值	说明
fingerPrint	指纹识别
facial	人脸识别（暂未支持）
speech	声纹识别（暂未支持）

示例代码如下所示：

```
wx.checkIsSupportSoterAuthentication({
      success(res) {
        // res.supportMode = [] 不具备任何被 SOTER 支持的生物识别方式
        // res.supportMode = ['fingerPrint'] 只支持指纹识别
        // res.supportMode = ['fingerPrint', 'facial'] 支持指纹识别和人脸识别
      }
    })
```

5.18　附近

小程序的附近 API，提供给需快速批量创建附近地点的小程序开发者使用，使用前请先调高附近地点额度。调高附近地点额度申请方式如下：

下载《调高地点额度申请表》，填写后发送至 placeofminiprogram@qq.com。邮件主题为：“账号名称”申请调高附近地点额度。审核通过后可调高额度。

1）添加地点

接口地址：

```
https://api.weixin.qq.com/wxa/addnearbypoi?access_token=ACCESS_ TOKEN
```

请求方式：POST。

请求参数说明如表 5-66 所示。

表 5-66　　添加地点请求参数说明

参数	说明	备注
related_name	经营资质主体	必填
related_credential	经营资质证件号	必填
related_address	经营资质地址	必填

添加请求暂不支持并发调用，建议使用时间隔 1s 进行串行调用。related_proof_material 通过临时素材上传接口上传，接口地址：

```
https://api.weixin.qq.com/cgi-bin/media/upload?access token=ACCESS TOKEN
&type=TYPE
```

临时素材［接口文档］详见微信公众平台-服务号-开发文档-素材管理-新增临时素材。

POST 数据示例代码如下所示：

```
{  "related_name":"×××公司",
"related_credential":"12345678-0",
"related_address":"广州市新港中路 397 号 TIT 创意园",  "related_proof_material":
"3LaLzqiTrQcD20DlX_o-OV1-nlYMu7sdVAL7SV2PrxVyjZFZZmB3O6LPGaYXlZWq"  }
```

返回参数说明，如表 5-67 所示。

表 5-67　　返回参数说明

参数	说明	备注
audit_id	审核单 ID	必填
poi_id	附近地点 ID	必填
related_credential	经营资质证件号	必填

返回内容示例如下所示：

```
{
"errcode":0,
"errmsg":"ok",
   "data":{
     "audit_id":416620525
}
}
```

错误码说明如表 5-68 所示。

表 5-68　　错误码说明

错误码	说明
47001	POST 数据 json 格式错误
20002	POST 参数非法
44002	POST 数据为空
92000	该经营资质已添加，请勿重复添加
92002	附近地点添加数量达到上限，无法继续添加
92004	附近功能被封禁
93011	个人类型小程序不可用
93012	非普通类型小程序（门店小程序、小店小程序等）不可用
93013	从腾讯地图获取地址详细信息失败
93014	同一资质证件号重复添加

事件推送，代码如下所示：

```
<xml>
 <ToUserName><![CDATA[gh_4346ac1514d8]]></ToUserName>
 <FromUserName><![CDATA[od1P50M-fNQI5Gcq-trm4a7apsU8]]></FromUserName>
```

非卖品，仅供非商业用途或交流学习使用

```
    <CreateTime>1488856741</CreateTime>
    <MsgType><![CDATA[event]]></MsgType>
    <Event><![CDATA[add_nearby_poi_audit_info]]></Event>
    <audit_id>11111</audit_id>
    <status>3</status>
    <reason><![CDATA[xxx]]></reason>
    <poi_id>111111</poi_id>
</xml>
```

参数说明如表 5-69 所示。

表 5-69　　参数说明

参数	说明
audit_id	审核单 id
status	审核状态（3：审核通过；2：审核失败）
reason	如果 status 为 3 或者 4，会返回审核失败的原因
poi_id	poi_id

2）查看地点列表

接口地址：

```
https://api.weixin.qq.com/wxa/getnearbypoilist?page=1&page_rows=20&access_token=ACCESS_TOKEN
```

请求方式: GET。

请求参数说明如表 5-70 所示。

表 5-70　　查看地点列表请求参数说明

参数	说明	备注
page	起始页 id（从 1 开始计数）	必填
page_rows	每页展示个数（最多 1000 个）	必填

返回参数说明如表 5-71 所示。

表 5-71　　返回参数说明

参数	说明	备注
data.left_apply_num	剩余可添加地点个数	
data.max_apply_num	最大可添加地点个数	
data.data	地址列表的 json 格式字符串	
data.data.poi_list[i].poi_id	poi_id	
data.data.poi_list[i].qualification_address	资质证件地址	
data.data.poi_list[i].qualification_num	资质证件证件号	
data.data.poi_list[i].audit_status	地点审核状态（3：审核中；4：审核失败；5：审核通过）	

非卖品，仅供非商业用途或交流学习使用

续表

参数	说明	备注
data.data.poi_list[i].display_status	地点展示在附近状态（0：未展示；1：展示中）	
data.data.poi_list[i].refuse_reason	审核失败原因	audit_status=4 时返回

返回 json 示例代码如下所示：

```
{
 "errcode":0,
 "errmsg":"",
 "data":
 {
    "left_apply_num" : 9,
    "max_apply_num" : 10,
    "data":
    "{
        "poi_list":
        [
           {
               "poi_id":"123456",
               "qualification_address":"广东省广州市海珠区新港中路 123 号",
               "qualification_num":"123456789-1",
               "audit_status":3,
               "display_status":0,
               "refuse_reason":""
           }
        ]
    }"
 }
}
```

错误返回码是-1，代表系统错误。

3）删除地点

接口地址：

```
https://api.weixin.qq.com/wxa/delnearbypoi?access_token=ACCESS_TOKEN
```

请求方式：POST。

请求参数说明如表 5-72 所示。

表 5-72　　删除地点请求参数说明

参数	说明	备注
poi_id	附近地点 ID	必填

POST 数据示例代码如下所示：

```
{
```

非卖品，仅供非商业用途或交流学习使用

```
  "poi_id":"469382092"
}
```

返回 json 示例代码如下所示：

```
{
  "errcode":92005,
  "errmsg":"地点正在审核中"
}
```

错误码说明如表 5-73 所示：

表 5-73　　错误码说明

错误码	说明
92004	附近功能被封禁
92005	地点正在审核中
92006	地点正在展示小程序
93010	地点不存在
93011	个人类型小程序不可用

4）展示/取消展示附近小程序

接口地址：

```
https://api.weixin.qq.com/wxa/setnearbypoishowstatus?access_token=ACCESS_TOKEN
```

请求方式：POST（请使用 https 协议）。

请求参数说明如表 5-74 所示。

表 5-74　　展示/取消展示附近小程序请求参数说明

参数	说明	备注
poi_id	附近地点 ID	必填
status	0：取消展示；1：展示	必填

POST 数据示例代码如下所示：

```
{
  "poi_id":"469487775",
  "status":0
}
```

返回 json 示例代码如下所示：

```
{
  "errcode":92008,
  "errmsg":"小程序未展示在该地点"
}
```

错误码说明如表 5-75 所示。

非卖品，仅供非商业用途或交流学习使用

表 5-75　　错误码说明

错误码	说明
92003	地点已被其他小程序占用
92004	附近功能被封禁
92005	地点正在审核中
92007	地点审核失败
92008	程序未展示在该地点
93009	小程序未上架或不可见
93010	地点不存在
93011	个人类型小程序不可用

5.19 插件管理

小程序开发者可通过以下接口管理插件，包括：添加、删除、查询已添加的插件。插件开发者也可调用接口管理插件使用申请。

1） 申请使用插件

此接口用于小程序向插件开发者发起使用插件的申请。

请求方式：POST（请使用 https 协议）。

```
https://api.weixin.qq.com/wxa/plugin?access_token=TOKEN
```

POST 数据示例代码如下所示：

```
{
  "action":"apply",
  "plugin_appid":"aaaa",
}
```

参数说明如表 5-76 所示。

表 5-76　　申请使用插件参数说明

参数	说明
action	填写 apply
plugin_appid	插件 appid

返回说明如下所示：

```
{
  "errcode":0,// 错误码
  "errmsg":"ok",//错误信息
}
```

错误码说明如表 5-77 所示。

非卖品，仅供非商业用途或交流学习使用

表 5-77　　错误码说明

错误码	说明
0	成功
-1	系统错误
89236	该插件不能申请
89237	已经添加该插件
89238	申请或使用的插件已经达到上限
89239	该插件不存在

2）查询已添加的插件

请求方式：POST（请使用 https 协议）：

```
https://api.weixin.qq.com/wxa/plugin?access_token=TOKEN
```

POST 数据示例如下所示：

```
{
  "action":"list",
}
```

参数说明如表 5-78 所示。

表 5-78　　查询已添加的插件参数说明

参数	说明
action	填写 list

返回说明（正常时返回的 json 示例）如下所示：

```
{
  "errcode":0,
  "errmsg":"ok",
  "plugin_list":[
  "appid":"aaaa",
  "status":1,
  "nickname":"插件昵称",
  "headimgurl":"http://plugin.qq.com",
  ],
}
```

返回参数说明如表 5-79 所示。

表 5-79　　返回参数说明

参数	说明
errcode	错误码
errmsg	错误信息
plugin_list	申请或使用中的插件列表
appid	插件 appid

非卖品，仅供非商业用途或交流学习使用

续表

参数	说明
status	插件状态（1：申请中；2：申请通过；3：已拒绝；4：已超时）
nickname	插件昵称
headimgurl	插件头像

错误码说明如表 5-80 所示。

表 5-80　　错误码说明

错误码	说明
0	成功
-1	系统错误
89243	“待确认”的申请不可删除
89044	不存在该插件 appid

3）删除已添加的插件

请求方式：POST（请使用 https 协议）。

```
https://api.weixin.qq.com/wxa/plugin?access_token=TOKEN
```

POST 数据示例代码如下所示：

```
{
  "action":"unbind" //填写 list
  "plugin_appid":"aaaa" //插件 appid
}
```

返回说明（正常时返回的 json 示例）如下所示：

```
{
  "errcode":0,
  "errmsg":"ok",
}
```

错误码说明如表 5-81 所示。

表 5-81　　错误码说明

错误码	说明
0	成功
-1	系统错误

4）获取当前所有插件使用方（供插件开发者调用）

请求方式：POST（请使用 https 协议）。

```
https://api.weixin.qq.com/wxa/devplugin?access_token=TOKEN
```

POST 数据示例如下所示：

非卖品，仅供非商业用途或交流学习使用

```
{
  "action":"dev_apply_list"
  "page":1
  "num":10
}
```

参数说明如表 5-82 所示。

表 5-82　　获取当前所有插件使用方的参数说明

参数	说明
action	填写 dev_apply_list
page、num	page 和 num 用于分页，表示每页 num 条记录，拉取第 page 页的数据

返回说明（正常时返回的 json 示例）如下所示：

```
{
  "errcode":0,
  "errmsg":"ok",
  "apply_list":[
  "appid":"aaaa",
  "status":1,
  "nickname":"小程序昵称",
  "headimgurl":"http://weixin.qq.com",
  ],
}
```

参数说明如表 5-83 所示。

表 5-83　　参数说明

参数	说明
apply_list	申请或使用中的插件列表
appid	使用者的 appid
status	插件状态（1：申请中；2：申请通过；3：已拒绝；4.已超时）
nickname	使用者的昵称
headimgurl	使用者的昵称

错误码说明如表 5-84 所示。

表 5-84　　错误码说明

错误码	说明
0	成功
-1	系统错误

5）修改插件使用申请的状态（供插件开发者调用）

请求方式：POST（请使用 https 协议）。

```
https://api.weixin.qq.com/wxa/devplugin?access_token=TOKEN
```

POST 数据示例代码如下所示：

非卖品，仅供非商业用途或交流学习使用

```
{
  "action":"dev_agree"  (或 dev_refuse、dev_delete)
}
```

参数说明如表 5-85 所示。

表 5-85　　修改插件使用申请状态的参数说明

参数	说明
action	dev_agree：同意申请；dev_refuse：拒绝申请；dev_delete：删除已拒绝的申请者
appid	使用者的 appid

返回说明（正常时返回的 json 示例）如下所示：

```
{
  "errcode":0,
  "errmsg":"ok",
}
```

错误码说明如表 5-86 所示。

表 5-86　　错误码说明

错误码	说明
0	成功
-1	系统错误
89240	无法进行此操作，只有“待确认”的申请可操作通过/拒绝
89241	无法进行此操作，只有“已拒绝/已超时”的申请可操作删除
89242	该 appid 不在申请列表内

非卖品，仅供非商业用途或交流学习使用

第六章

数据、更新、多线程

本章主要介绍数据统计分析、小程序版本更新，以及多线程的使用。

6.1 数据

开发者通过数据分析接口，可获取到小程序的各项数据指标，便于进行数据存储和整理。用户访问小程序的详细数据可从访问分析中获取，概况中提供累计用户数等部分指标数据。

6.1.1 概况趋势

接口地址：

```
https://api.weixin.qq.com/datacube/getweanalysisappiddailysummarytrend?access_token=ACCESS_TOKEN
```

POST 请求参数说明如表 6-1 所示。

表 6-1　　POST 请求参数说明

参数	是否必填	说明
begin_date	是	开始日期
end_date	是	结束日期，限定查询 1 天数据，end_date 允许设置的最大值为昨日

POST 内容示例代码如下所示：

```
{
  "begin_date" : "20170313",
    "end_date" : "20170313"
}
```

返回参数说明如表 6-2 所示。

表 6-2　　返回参数说明

参数	说明
visit_total	累计用户数

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

续表

参数	说明
share_pv	转发次数
share_uv	转发人数

返回数据示例代码如下所示：

```
{
  "list": [
     {
        "ref_date": "20170313",
        "visit_total": 391,
        "share_pv": 572,
        "share_uv": 383
     }
  ]
}
```

6.1.2 访问趋势

1）日趋势

接口地址：

```
https://api.weixin.qq.com/datacube/getweanalysisappiddailyvisittrend?access_token=ACCESS_TOKEN
```

POST 参数说明如表 6-3 所示。

表 6-3 POST 参数说明

参数	是否必填	说明
begin_date	是	开始日期
end_date	是	结束日期，限定查询 1 天数据，end_date 允许设置的最大值为昨日

POST 内容示例代码如下所示：

```
{
  "begin_date" : "20170313",
   "end_date" : "20170313"
}
```

返回参数说明如表 6-4 所示。

表 6-4 返回参数说明

参数	说明
ref_date	时间，如："20170313"
session_cnt	打开次数
visit_pv	访问次数

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

续表

参数	说明
visit_uv	访问人数
visit_uv_new	新用户数
stay_time_uv	人均停留时长（浮点型，单位为 s）
stay_time_session	次均停留时长（浮点型，单位为 s）
visit_depth	平均访问深度（浮点型）

返回数据示例代码如下所示：

```
{
  "list": [
    {
      "ref_date": "20170313",
      "session_cnt": 142549,
      "visit_pv": 472351,
      "visit_uv": 55500,
      "visit_uv_new": 5464,
      "stay_time_session": 0,
      "visit_depth": 1.9838
    }
  ]
}
```

2）周趋势

接口地址：

```
https://api.weixin.qq.com/datacube/getweanalysisappidweeklyvisittrend?
access_token=ACCESS_TOKEN
```

POST 参数说明如表 6-5 所示。

表 6-5　　POST 参数说明

参数	是否必填	说明
begin_date	是	开始日期，为周一日期
end_date	是	结束日期，为周日日期，限定查询一周数据

请求 json 和返回 json 与天的一致，这里限定查询一个自然周的数据，时间必须按照自然周的方式输入，如：20170306（周一），20170312（周日）。

POST 内容示例代码如下所示：

```
{
  "begin_date":"20170306",
    "end_date":"20170312"
}
```

返回参数说明如表 6-6 所示。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 6-6 返回参数说明

参数	说明
ref_date	时间范围，如："20170306-20170312"
session_cnt	打开次数（自然周内汇总）
visit_pv	访问次数（自然周内汇总）
visit_uv	访问人数（自然周内去重）
visit_uv_new	新用户数（自然周内去重）
stay_time_uv	人均停留时长（浮点型，单位为 s）
stay_time_session	次均停留时长（浮点型，单位为 s）
visit_depth	平均访问深度（浮点型）

返回内容示例代码如下所示：

```
{
  "list": [
     {
        "ref_date": "20170306-20170312",
        "session_cnt": 986780,
        "visit_pv": 3251840,
        "visit_uv": 189405,
        "visit_uv_new": 45592,
        "stay_time_session": 54.5346,
        "visit_depth": 1.9735
     }
  ]
}
```

3）月趋势

接口地址：

```
https://api.weixin.qq.com/datacube/getweanalysisappidmonthlyvisittrend?access_token=ACCESS_TOKEN
```

POST 参数说明如表 6-7 所示。

表 6-7 POST 参数说明

参数	是否必填	说明
begin_date	是	开始日期，为自然月第一天
end_date	是	结束日期，为自然月最后一天，限定查询一个月数据

请求 json 和返回 json 与天的一致，这里限定查询一个自然月的数据，时间必须按照自然月的方式输入，如：20170201（月初），20170228（月末）。

POST 内容示例代码如下所示：

```
{
  "begin_date":"20170201",
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    "end_date":"20170228"
}
```

返回参数说明如表 6-8 所示。

表 6-8　　返回参数说明

参数	说明
ref_date	时间，如："201702"
session_cnt	打开次数（自然月内汇总）
visit_pv	访问次数（自然月内汇总）
visit_uv	访问人数（自然月内去重）
visit_uv_new	新用户数（自然月内去重）
stay_time_uv	人均停留时长（浮点型，单位为 s）
stay_time_session	次均停留时长（浮点型，单位为 s）
visit_depth	平均访问深度（浮点型）

返回内容示例代码如下所示：

```
{
  "list": [
    {
      "ref_date": "201702",
      "session_cnt": 126513,
      "visit_pv": 426113,
      "visit_uv": 48659,
      "visit_uv_new": 6726,
      "stay_time_session": 56.4112,
      "visit_depth": 2.0189
    }
  ]
}
```

6.1.3　访问分布

接口地址：

```
    https://api.weixin.qq.com/datacube/getweanalysisappidvisitdistribution?
access_token=ACCESS_TOKEN
```

POST 参数说明如表 6-9 所示。

表 6-9　　POST 参数说明

参数	是否必填
begin_date	是
end_date	是

POST 内容示例如下页所示：

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
{
  "begin_date":"20170313",
    "end_date":"20170313"
}
```

返回参数说明如表 6-10 所示。

表 6-10 返回参数说明

参数	说明
ref_date	时间，如："20170313"
list	存入所有类型的指标情况

list 的每一项包括值如表 6-11 所示。

表 6-11 list 的每一项包括值

参数	说明
index	分布类型
item_list	分布数据列表

分布类型（index）的取值范围如表 6-12 所示。

表 6-12 分布类型（index）的取值范围

参数	说明
access_source_session_cnt	访问来源分布
access_staytime_info	访问时长分布
access_depth_info	访问深度的分布

每个数据项包括值如表 6-13 所示。

表 6-13 每个数据项包括值

参数	说明
key	场景 id
value	场景下的值（均为整数型）

key 对应关系如下所示。

访问来源：(index="access_source_session_cnt")
1：小程序历史列表
2：搜索
3：会话
4：二维码
5：公众号主页

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

6：聊天顶部
7：系统桌面
8：小程序主页
9：附近的小程序
10：其他
11：模板消息
12：客服消息
13：公众号菜单
14：App 分享
15：支付完成页
16：长按识别二维码
17：相册选取二维码
18：公众号文章
19：钱包
20：卡包
21：小程序内卡券
22：其他小程序
23：其他小程序返回
24：卡券适用门店列表
25：搜索框快捷入口
26：小程序客服消息
27：公众号下发
28：会话左下角菜单
29：小程序任务栏
30：长按小程序菜单圆点
31：连 Wi-Fi 成功页
32：城市服务
访问时长：(index="access_staytime_info")
1: 0～2s
2: 3～5s
3: 6～10s
4: 11～20s
5: 20～30s

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

6: 30～50s
7: 50～100s
8: > 100s
平均访问深度：(index="access_depth_info")
1: 1 页
2: 2 页
3: 3 页
4: 4 页
5: 5 页
6: 6～10 页
7: >10 页

返回数据示例代码如下所示：

```
{
  "ref_date": "20170313",
    "list": [
      {
        "index": "access_source_session_cnt",
        "item_list": [
          {
            "key": 10,
            "value": 5
          },
          {
            "key": 8,
            "value": 687
          },
          {
            "key": 7,
            "value": 10740
          },
          {
            "key": 6,
            "value": 1961
          },
          {
            "key": 5,
            "value": 677
          },
          {
            "key": 4,
            "value": 653
          },
          {
            "key": 3,
            "value": 1120
          },
          {
            "key": 2,
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
                "value": 10243
            },
            {
                "key": 1,
                "value": 116578
            }
        ]
    },
    {
        "index": "access_staytime_info",
        "item_list": [
            {
                "key": 8,
                "value": 16329
            },
            {
                "key": 7,
                "value": 19322
            },
            {
                "key": 6,
                "value": 21832
            },
            {
                "key": 5,
                "value": 19539
            },
            {
                "key": 4,
                "value": 29670
            },
            {
                "key": 3,
                "value": 19667
            },
            {
                "key": 2,
                "value": 11794
            },
            {
                "key": 1,
                "value": 4511
            }
        ]
    },
    {
        "index": "access_depth_info",
        "item_list": [
            {
                "key": 5,
                "value": 217
            },
            {
                "key": 4,
                "value": 3259
            },
            {
```


非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

```
                "key": 3,
                "value": 32445
            },
            {
                "key": 2,
                "value": 63542
            },
            {
                "key": 1,
                "value": 43201
            }
        ]
    }
]
}
```

6.1.4 访问留存

1）日留存

接口地址：

```
https://api.weixin.qq.com/datacube/getweanalysisappiddailyretaininfo?
access_token=ACCESS_TOKEN
```

POST 参数说明如表 6-14 所示。

表 6-14 POST 参数说明

参数	是否必填	说明
begin_date	是	开始日期
end_date	是	结束日期，限定查询 1 天数据，end_date 允许设置的最大值为昨日

POST 内容示例代码如下所示：

```
{
  "begin_date" : "20170313",
    "end_date" : "20170313"
}
```

返回参数说明如表 6-15 所示。

表 6-15 返回参数说明

参数	说明
visit_uv_new	新增用户留存
visit_uv	活跃用户留存

visit_uv、visit_uv_new 的每一项包括参数 key 和 value，如表 6-16 所示。

非卖品，仅供非商业用途或交流学习使用

表 6-16　　visit_uv、visit_uv_new 的每一项包含的参数

参数	说明
key	标识，从 0 开始，0 表示当天，1 表示 1 天后，依此类推，key 取值分别是：0、1、2、3、4、5、6、7、14、30
value	key 对应日期的新增用户数/活跃用户数（key=0 时）或留存用户数（k>0 时）

返回数据示例代码如下所示：

```
{
  "ref_date": "20170313",
    "visit_uv_new": [
      {
         "key": 0,
         "value": 5464
      }
    ],
      "visit_uv": [
         {
           "key": 0,
           "value": 55500
         }
      ]
}
```

2）周留存

接口地址：

```
https://api.weixin.qq.com/datacube/getweanalysisappidweeklyretaininfo?access_token=ACCESS_TOKEN
```

POST 参数说明如表 6-17 所示。

表 6-17　　POST 参数说明

参数	是否必填	说明
begin_date	是	开始日期，为周一日期
end_date	是	结束日期，为周日日期，限定查询一周数据

请求 json 和返回 json 与天的一致，这里限定查询一个自然周的数据，时间必须按照自然周的方式输入，如：20170306（周一），20170312（周日）。

POST 内容示例如下所示：

```
{
  "begin_date" : "20170306",
    "end_date" : "20170312"
}
```

返回参数说明如表 6-18 所示。

非卖品，仅供非商业用途或交流学习使用

表 6-18 返回参数说明

参数	说明
ref_date	时间范围，如："20170306-20170312"
visit_uv_new	新增用户留存
visit_uv	活跃用户留存

visit_uv、visit_uv_new 的每一项包括参数 key 和 value，如表 6-19 所示。

表 6-19 visit_uv、visit_uv_new 的每一项包含的参数

参数	说明
key	标识，从 0 开始，0 表示当周，1 表示 1 周后，依此类推，key 取值分别是：0、1、2、3、4
value	key 对应日期的新增用户数/活跃用户数（key=0 时）或留存用户数（k>0 时）

返回数据示例代码如下所示：

```
{
  "ref_date": "20170306-20170312",
   "visit_uv_new": [
      {
         "key": 0,
         "value": 0
      },
      {
         "key": 1,
         "value": 16853
      }
   ],
      "visit_uv": [
         {
            "key": 0,
            "value": 0
         },
         {
            "key": 1,
            "value": 99310
         }
      ]
}
```

3）月留存

接口地址：

```
    https://api.weixin.qq.com/datacube/getweanalysisappidmonthlyretaininfo?
access_token=ACCESS_TOKEN
```

POST 参数说明如表 6-20 所示。

表 6-20 POST 参数说明

参数	是否必填	说明
begin_date	是	开始日期，为自然月第一天
end_date	是	结束日期，为自然月最后一天，限定查询一个月数据

非卖品，仅供非商业用途或交流学习使用

请求 json 和返回 json 与天的一致，这里限定查询一个自然月的数据，时间必须按照自然月的方式输入，如：20170201（月初），20170228（月末）。

POST 内容示例代码如下：

```
{
  "begin_date":"20170201",
   "end_date":"20170228"
}
```

返回参数说明如表 6-21 所示。

表 6-21　　返回参数说明

参数	说明
ref_date	时间，如："201702"
visit_uv_new	新增用户留存
visit_uv	活跃用户留存

visit_uv、visit_uv_new 的每一项包括参数 key 和 value，如表 6-22 所示。

表 6-22　　visit_uv、visit_uv_new 的每一项包含的参数

参数	说明
key	标识，从 0 开始，0 表示当月，1 表示 1 月后，key 取值分别是：0、1
value	key 对应日期的新增用户数/活跃用户数（key=0 时）或留存用户数（k>0 时）

返回内容示例代码如下所示：

```
{
  "ref_date": "201702",
    "visit_uv_new": [
       {
          "key": 0,
          "value": 346249
       }
    ],
       "visit_uv": [
          {
             "key": 0,
             "value": 346249
          }
       ]
}
```

6.1.5　访问页面

目前小程序只提供按 page_visit_pv 排序的 top200。

接口地址：

```
    https://api.weixin.qq.com/datacube/getweanalysisappidvisitpage?access_token=
ACCESS_TOKEN
```

非卖品，仅供非商业用途或交流学习使用

POST 参数说明如表 6-23 所示。

表 6-23 POST 参数说明

参数	是否必填	说明
begin_date	是	开始日期
end_date	是	结束日期，限定查询 1 天数据，end_date 允许设置的最大值为昨日

POST 内容示例代码如下所示：

```
{
  "begin_date":"20170313",
    "end_date":"20170313"
}
```

返回参数说明如表 6-23 所示。

表 6-23 返回参数说明

参数	说明
page_path	页面路径
page_visit_pv	访问次数
page_visit_uv	访问人数
page_staytime_pv	次均停留时长
entrypage_pv	进入页次数
exitpage_pv	退出页次数
page_share_pv	转发次数
page_share_uv	转发人数

返回内容示例代码如下所示：

```
{
  "ref_date": "20170313",
    "list": [
       {
          "page_path": "pages/main/main.html",
          "page_visit_pv": 213429,
          "page_visit_uv": 55423,
          "page_staytime_pv": 8.139198,
          "entrypage_pv": 117922,
          "exitpage_pv": 61304,
          "page_share_pv": 180,
          "page_share_uv": 166
       },
       {
          "page_path": "pages/linedetail/linedetail.html",
          "page_visit_pv": 155030,
          "page_visit_uv": 42195,
          "page_staytime_pv": 35.462395,
```

非卖品，仅供非商业用途或交流学习使用

```
            "entrypage_pv": 21101,
            "exitpage_pv": 47051,
            "page_share_pv": 47,
            "page_share_uv": 42
        },
        {
            "page_path": "pages/search/search.html",
            "page_visit_pv": 65011,
            "page_visit_uv": 24716,
            "page_staytime_pv": 6.889634,
            "entrypage_pv": 1811,
            "exitpage_pv": 3198,
            "page_share_pv": 0,
            "page_share_uv": 0
        },
        {
            "page_path": "pages/stationdetail/stationdetail.html",
            "page_visit_pv": 29953,
            "page_visit_uv": 9695,
            "page_staytime_pv": 7.558508,
            "entrypage_pv": 1386,
            "exitpage_pv": 2285,
            "page_share_pv": 0,
            "page_share_uv": 0
        },
        {
            "page_path": "pages/switch-city/switch-city.html",
            "page_visit_pv": 8928,
            "page_visit_uv": 4017,
            "page_staytime_pv": 9.22659,
            "entrypage_pv": 748,
            "exitpage_pv": 1613,
            "page_share_pv": 0,
            "page_share_uv": 0
        }
    ]
}
```


返回参数说明如表 6-24 所示。

表 6-24　　返回参数说明

参数	说明
page_path	页面路径
page_visit_pv	访问次数
page_visit_uv	访问人数
page_staytime_pv	次均停留时长
entrypage_pv	进入页次数
exitpage_pv	退出页次数
page_share_pv	转发次数
page_share_uv	转发人数

非卖品，仅供非商业用途或交流学习使用

6.1.6 用户画像

获取小程序新增或活跃用户的画像分布数据。时间范围支持昨天、最近 7 天、最近 30 天。其中，新增用户数为时间范围内首次访问小程序的去重用户数，活跃用户数为时间范围内访问过小程序的去重用户数。画像属性包括用户年龄、性别、省份、城市、终端类型、机型。

接口地址：

```
https://api.weixin.qq.com/datacube/getweanalysisappiduserportrait?access_token=ACCESS_TOKEN
```

POST 请求参数说明如表 6-25 所示。

表 6-25 POST 请求参数说明

参数	是否必填	说明
begin_date	是	开始日期
end_date	是	结束日期，开始日期与结束日期相差的天数限定为 0/6/29，分别表示查询最近 1/7/30 天数据，end_date 允许设置的最大值为昨日

POST 内容示例代码如下所示：

```
{
  "begin_date" : "2017-06-11",
   "end_date" : "2017-06-17"
}
```

每次请求返回选定的时间范围及以下指标项，如表 6-26 所示。

表 6-26 返回指标

参数	说明
ref_date	时间范围，如："20170611-20170617"
visit_uv_new	新用户
visit_uv	活跃用户

每个指标项下包括的属性，如表 6-27 所示。

表 6-27 指标属性

参数	说明
province	省份，如河北、广东等
city	城市，如北京、广州等
genders	性别，包括男、女、未知
platforms	终端类型，包括 iPhone、android、其他
devices	机型，如苹果 iPhone6、OPPO R9 等
ages	年龄，包括 17 岁以下、18～24 岁等区间

非卖品，仅供非商业用途或交流学习使用

每个属性下包括的数据项，如表 6-28 所示。

表 6-28　　属性的数据项

参数	说明
id	属性值 id
name	属性值名称，与 id 一一对应。如属性为 province 时，返回的属性值名称包括“广东省”等
value	属性值对应的指标值，如指标为 visit_uv，属性为 province，属性值为“广东省”，value 对应广东地区的活跃用户

由于部分用户属性数据缺失，属性值可能出现“未知”。机型数据无 id 字段，暂时只提供用户数最多的 top20。返回数据示例代码如下所示：

```
{
  "ref_date": "20170611",
    "visit_uv_new": {
    "province": [
      {
        "id": 31,
        "name": "广东省",
        "value": 215
      }
    ],
      "city": [
        {
          "id": 3102,
          "name": "广州",
          "value": 78
        }
      ],
        "genders": [
          {
            "id": 1,
            "name": "男",
            "value": 2146
          }
        ],
          "platforms": [
            {
              "id": 1,
              "name": "iPhone",
              "value": 27642
            }
          ],
            "devices": [
              {
                "name": "OPPO R9",
                "value": 61
              }
            ],
              "ages": [
                {
```

非卖品，仅供非商业用途或交流学习使用

```
                    "id": 1,
                    "name": "17岁以下",
                    "value": 151
                }
            ]
    },
    "visit_uv": {
      "province": [
        {
          "id": 31,
          "name": "广东省",
          "value": 1341
        }
      ],
        "city": [
          {
             "id": 3102,
             "name": "广州",
             "value": 234
          }
        ],
          "genders": [
            {
              "id": 1,
              "name": "男",
              "value": 14534
            }
          ],
            "platforms": [
              {
                 "id": 1,
                 "name": "iPhone",
                 "value": 21750
              }
            ],
              "devices": [
                {
                   "name": "OPPO R9",
                   "value": 617
                }
              ],
                "ages": [
                  {
                     "id": 1,
                     "name": "17岁以下",
                     "value": 3156
                  }
                ]
    }
}
```


6.1.7 自定义分析

wx.reportAnalytics(eventName, data)：自定义分析数据上报接口。使用前，需要在小

非卖品，仅供非商业用途或交流学习使用

程序管理后台自定义分析中新建事件，配置好事件名与字段。

参数说明如表 6-29 所示。

表 6-29 参数说明

参数	类型	必填	说明
eventName	String	是	事件名
data	Object	是	上报的自定义数据。key 为配置中的字段名，value 为上报的数据

使用代码如下所示：

```
wx.reportAnalytics('purchase', {
  price: 120,
  color: 'red'
})
```

6.2 更新

从基础库版本 1.9.90 开始支持，获取全局唯一的版本更新管理器，用于管理小程序更新。

运行机制

小程序启动会有两种情况：一种是热启动，另一种是冷启动。假如用户已经打开过某个小程序，然后在一定时间内再次打开该小程序，此时无须重新启动，只需将后台的小程序切换到前台，这个过程就是热启动；冷启动指的是用户首次打开或小程序被微信主动销毁后再次打开的情况，此时小程序需要重新加载启动。

小程序没有重启的概念，当小程序进入后台，客户端会维持一段时间的运行状态，超过一定时间后（目前是 5 min）会被微信主动销毁。当短时间内（5s）连续两次以上收到系统内存告警，微信会进行小程序的销毁。

更新机制

小程序冷启动时，如果发现有新版本，将会异步下载新版本的代码包，并同时用客户端的本地包进行启动，即新版本的小程序需要等下一次冷启动时才会应用上。如果需要马上应用最新版本，可以使用 wx.getUpdateManager API 进行处理。

再次打开逻辑

从基础库版本 1.4.0 开始支持，用户打开小程序预期有以下两类场景：

A. 打开首页：场景值有 1001、1019、1022、1023、1038、1056。

B. 打开小程序指定的某个页面：场景值为除 A 以外的其他场景值。

再次打开一个小程序逻辑如表 6-30 所示。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

表 6-30 再次打开一个小程序逻辑

上一次的场景	当前打开的场景	效果
A	A	保留原来的状态
B	A	清空原来的页面栈，打开首页（相当于执行 wx.reLaunch 到首页）
A 或 B	B	清空原来的页面栈，打开指定页面（相当于执行 wx.reLaunch 到指定页）

使用 API：wx.getUpdateManager()来更新小程序，updateManager 对象的方法如表 6-31 所示。

表 6-31 updateManager 对象的方法

方法	参数	说明
onCheckForUpdate	callback	当向微信后台请求完新版本信息，会进行回调
onUpdateReady	callback	当新版本下载完成，会进行回调
onUpdateFailed	callback	当新版本下载失败，会进行回调
applyUpdate		当新版本下载完成，调用该方法会强制当前小程序应用新版本并重启

onCheckForUpdate(callback) 回调结果说明如表 6-32 所示。

表 6-32 onCheckForUpdate(callback) 回调结果说明

属性	类型	说明
hasUpdate	Boolean	是否有新的版本

检查更新操作由微信在小程序冷启动时自动触发，不需由开发者主动触发，开发者只需监听检查结果即可。

onUpdateReady(callback) 回调结果说明：当微信检查到小程序有新版本，会主动触发下载操作（无须开发者触发），当下载完成后，会通过 onUpdateReady 告知开发者。

onUpdateFailed(callback) 回调结果说明：当微信检查到小程序有新版本，会主动触发下载操作（无须开发者触发），如果下载失败（可能是网络原因等），会通过 onUpdateFailed 告知开发者。

applyUpdate() 说明：当小程序新版本已经下载时（即收到 onUpdateReady 回调），开发者可以通过这个方法强制重启小程序并应用最新版本。

示例代码如下所示：

```
  const updateManager = wx.getUpdateManager()

updateManager.onCheckForUpdate(function (res) {
    // 请求完新版本信息的回调
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    console.log(res.hasUpdate)
  })

  updateManager.onUpdateReady(function () {
    wx.showModal({
      title: '更新提示',
      content: '新版本已经准备好，是否重启应用？',
      success: function (res) {
        if (res.confirm) {
          // 新的版本已经下载好，调用 applyUpdate 应用新版本并重启
          updateManager.applyUpdate()
        }
      }
    })

  })

  updateManager.onUpdateFailed(function () {
    // 新的版本下载失败
  })
```

6.3 多线程

从基础库版本 1.9.90 开始支持多线程 API：wx.createWorker(scriptPath)。使用 createWorker 来创建一个 Worker 线程，并返回 Worker 实例。

一些异步处理的任务可以放置于 Worker 中运行，待运行结束后，再把结果返回到小程序主线程。Worker 运行于一个单独的全局上下文与线程中，不能直接调用主线程的方法。Worker 与主线程之间的数据传输，双方使用 Worker.postMessage() 来发送数据，Worker.onMessage() 来接收数据，传输的数据并不是直接共享，而是被复制的。

Worker 使用步骤

1. 配置 Worker 信息

在 app.json 中可配置 Worker 代码放置的目录，该目录下的代码将被打包成一个文件。

配置代码如下所示：

```
{
    "workers": "workers"
}
```

2. 添加 Worker 代码文件

根据步骤 1 设置的文件夹名称，在代码目录下创建该文件“workers”，并在 workers 文件夹下面创建“request”和“response”文件夹。在这两个文件夹下创建以下 3 个 js 文件：

```
workers/request/index.js
workers/request/utils.js
workers/response/index.js
```

添加完成之后，目录结构如图 6-1 所示。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

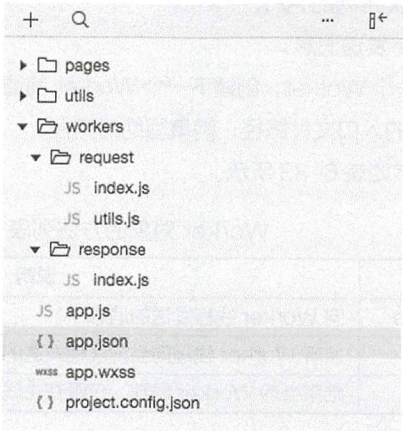


图 6-1 目录结构

3. 编写 Worker 代码

在 workers/request/index.js 编写 Worker 响应代码，如下所示：

```
var utils = require('./utils')

// 在 Worker 线程执行上下文会全局暴露一个 'worker' 对象，直接调用 worker.onMeesage/
// postMessage 即可
worker.onMessage(function (res) {
  console.log(res)
})
```

4. 在主线程中初始化 Worker

在主线程的 app.js 代码中初始化 Worker，代码如下所示：

```
//app.js
var worker = wx.createWorker('workers/request/index.js') // 文件名指定 Worker
//的入口文件路径，即绝对路径
```

5. 主线程向 Worker 发送消息

代码如下所示：

```
  worker.postMessage({
    msg: 'hello worker'
  })
```

注意事项：

1. Worker 最大并发数量限制为 1 个，创建下一个前请用 Worker.terminate()结束当前 Worker；

2. Worker 内代码只能用 require 指定 Worker 路径内的文件，无法引用其他路径；

3. Worker 的入口文件由 wx.createWorker()指定时，开发者可动态指定 Worker 入口文件；

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

4. Worker 内不支持 wx 系列的 API；

5. Workers 之间不支持发送消息。

目前限制最多只能创建一个 Worker，创建下一个 Worker 前请调用 Worker.terminate。

scriptPath 为 Worker 的入口文件路径，需填写绝对路径。

Worker 对象的方法列表如表 6-33 所示。

表 6-33　　Worker 对象的方法列表

方法	参数	说明
postMessage	message	向 Worker 线程发送的消息
onMessage	callback	监听 Worker 线程向当前线程发送的消息
terminate		结束当前 Worker 线程，仅限在主线程 Worker 实例上调用

postMessage(message)说明：向 Worker 线程发送消息，message 参数为需要发送的消息，必须是一个可序列化的 JavaScript 对象。

onMessage(callback) 回调结果说明如表 6-34 所示。

表 6-34　　onMessage(callback) 回调结果说明

属性	类型	说明
message	Object	Worker 线程向当前线程发送的消息

terminate()说明：结束当前 Worker 线程，仅限在主线程 Worker 对象上调用。

示例代码：运行以下代码需先进行基础配置，详细请查阅多线程文档，了解基础知识和配置方法。

```
const worker = wx.createWorker('workers/request/index.js') // 文件名指定
//Worker 的入口文件路径，即绝对路径

worker.onMessage(function (res) {
  console.log(res)
})
worker.postMessage({
  msg: 'hello worker'
})
worker.terminate()
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

第七章

项目实战

本书前面 6 章基本涵盖了微信小程序开发的所有知识点，不管是新闻类、阅读类、娱乐、游戏等小应用，都是利用前面的各种知识点来进行开发。

本章将以几个实际项目案例来讲解小程序的开发过程和代码实现，带领大家从 0 到 1 实现自己的小程序。

本章将分别介绍 6 个小应用案例。

- 仿新闻小应用：实现不同分类（热点、社会、娱乐、科技、体育等）的新闻列表。
- 仿电子书小应用：实现书店类应用展示，通过几种不同样式去布局展示。
- 录音功能：按住录音并保存，可以播放历史录音记录。
- 二维码生成器：利用开源 js 去实现一些常用功能，帮助自己减少开发难度。
- 图片滤镜：选取图片或拍照。实现几种常见的图片滤镜效果。
- 仿电影小应用：实现电影列表，电影详情、标签热词搜索等功能。

7.1 仿新闻小应用

在互联网如此发达的今天，信息大爆炸的时代，新闻类应用不容忽视，除了内容的及时和丰富，更重要的是通过丰富的 UI 展示样式去提高用户阅读体验，才可以保持留存率，防止用户流失。

本节主要介绍新闻类应用顶部滑动菜单的实现、新闻列表的实现以及实现用户中心的复杂布局。

项目目录如图 7-1 所示。

项目显示效果如图 7-2 所示。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

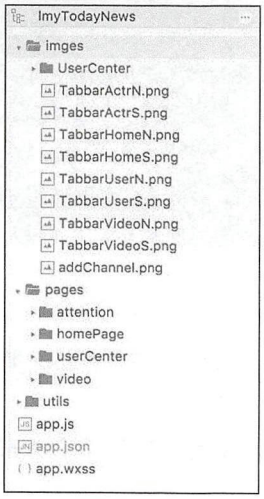


图 7-1 新闻类应用项目结构

图 7-2 新闻类应用展示效果

7.1.1 通过 tabBar 实现页面之间的切换

使用 tabBar 功能添加 4 个页面，实现“首页”“视频”“关注”“我的”主题结构。代码如下所示：

```
{
  "pages":[
    "pages/homePage/homePage",
    "pages/attention/attention",
    "pages/userCenter/userCenter",
    "pages/video/video"
  ],
  "window":{
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#ff5400",
    "navigationBarTitleText": "新闻",
    "navigationBarTextStyle":"white"
  },
   "tabBar": {
    "color": "#666666",
    "selectedColor": "#ffa589",
    "backgroundColor": "#eeeeee",
    "borderStyle": "white",
    "list": [{
      "pagePath": "pages/homePage/homePage",
      "iconPath":"imges/TabbarHomeN.png",
      "selectedIconPath":"imges/TabbarHomeS.png",
      "text": "首页"
    }, {
      "pagePath": "pages/video/video",
      "iconPath":"imges/TabbarVideoN.png",
      "selectedIconPath":"imges/TabbarVideoS.png",
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        "text": "视频"
      },{
        "pagePath": "pages/attention/attention",
        "iconPath":"imges/TabbarActrN.png",
        "selectedIconPath":"imges/TabbarActrS.png",
        "text": "关注"
      },{
        "pagePath": "pages/userCenter/userCenter",
        "iconPath":"imges/TabbarUserN.png",
        "selectedIconPath":"imges/TabbarUserS.png",
        "text": "我的"
      }]
    }
  }
```

请注意，tabBar 数组使用到的页面必须存在，也就是说在“pages”数组中。不然 tabBar 显示会出现问题。

```
color: 标题颜色
selectedColor: 选中之后的标题颜色
backgroundColor: 背景颜色
borderStyle: 风格
pagePath: 页面的路径
iconPath: icon 图标
selectedIconPath: 选中之后的 icon 图标
text: 标题文本
```

7.1.2 顶部滑动菜单的实现

使用一个横向滚动的 scroll-view 来实现顶部滑动菜单，代码如下所示：

```
    <!--顶部 分类栏目-->
    <view class="topTabBarClass">

      <!--类型 滚动视图-->
      <scroll-view scroll-x="true" style="width: 90%;height: 100%; white-
space: nowrap;display: flex">
        <block wx:for="{{tArray}}">
          <view animation="{{animation}}" class="topTypeListClass {{index==
curpage? 'curPage':''}}" bindtap="typeClick" id="{{index}}" data-idx="{{item.
category}}">
            {{item.name}}
          </view>
        </block>
      </scroll-view>

      <!--右边按钮-->
      <view class="rightview">
        <image mode="aspectFit" src="../../imges/addChannel.png" class=
"addImage"> </image>
      </view>
    </view>
```

block 模块通过 wx:for 读取 tArray 数组内容来创建 view 控件，animation 绑定动画，

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

bindtap 绑定单击事件，data-idx 绑定单击事件的参数。view 组件显示的内容是 item.name。

在 js 文件中，页面加载的时候去请求网络，获取所有类型，代码如下所示：

```
// 总接口
let ApiUrl = "http://wx.leadingdo.com/demo/"

// 新闻类型接口
let tyepUrl = ApiUrl + "news/type.aspx"

// 页面加载
  onLoad: function () {

    //导航栏显示加载状态
    wx.showNavigationBarLoading();

    // 定义 this 代理，处理网络返回数据，不能直接使用 this
    var that = this;

    //请求网络，获取 type
    wx.request({
      url: tyepUrl,
      data: {
      },
      header: {
        'content-type': 'application/json'
      },
      success: function (res) {

        //获取返回的数组
        let dataArr = [];
        dataArr = res.data.data.data;

        //打印输出
        console.log(dataArr);

        //变量赋值
        that.setData({
          tArray: dataArr
        });
      }, fail: function (res) {

      }, complete: function (res) {

        //取消导航栏加载
        wx.hideNavigationBarLoading();

      }
    })

    //初始化页码从 0 开始
    this.setData({
      listpage: 0
    });

    //默认显示所有新闻
    this.readList("all")
  },
```

用到的 CSS 样式如下所示：

```
/*顶部分类栏目*/

.topTabBarClass {
  background-color: #ffa589;
  display: flex;
  flex-direction: row;
  width: 100%;
  height: 80rpx;
}

/*类型选择 默认样式*/
.topTypeListClass {
  display: inline-block;
  margin-top: 12rpx;
  margin-left: 10rpx;
  font-size: 30rpx;
  width: 100rpx;
  height: 100%;
  text-align: center;
  z-index: 99;
  color: #333;
}
```

分类按钮事件，代码如下所示：

```
// 类型单击事件
  typeClick(e) {

    var idx = e.currentTarget.dataset.idx;
    console.log(idx);

    var that = this;
    that.setData({
      curpage: e.target.id
    });

    //初始化页码从 0 开始
    this.setData({
      listpage: 0
    });

    console.log("curpage=", this.data.curpage);
    console.log("listpage==", this.data.listpage);

    //设置分类
    this.setData({
      category: idx
    });

    //获取新闻
    this.readList()
  },
```

小程序通过 e.currentTarget.dataset.idx 来获取单击的项目，用此类型重新请求网络获取新闻列表内容。

非卖品，仅供非商业用途或交流学习使用

7.1.3 新闻列表的实现

新闻列表首先需要请求网络，获取新闻，关键代码如下所示：

```
// 新闻列表接口
let newsUrl = ApiUrl + "news/list.aspx"
//获取新闻列表
  readList() {

    // // 显示加载状态
    this.setData({
      loading: false
    })

    //请求网络，获取 type
    var that = this;
    wx.request({
      url: newsUrl,
      method: "POST",//默认 GET
      data: {
        "category": this.data.category,
        "page": this.data.curpage
      },
      header: {

        //以表单形式提交
        "content-type": "application/x-www-form-urlencoded"

        //以 json 形式提交
        //  "content-type": "application/json"
      },
      success: function (res) {

        //如果是第 1 页，坐标是 0，数组先清空
        if (that.data.listpage == 0) {
          that.setData({
            detaildata: [],
          });
        }

        //解析数据
        var arr = res.data.data;
        console.log(arr)
        var dataArr = [];
        for (var index of arr) {
          //转换 Json 字符串为 Json 对象
          dataArr.push(JSON.parse(index.content));
        }

        //修改数组
        that.setData({
          detaildata: dataArr,
        });
```

非卖品，仅供非商业用途或交流学习使用

```
        }, fail: function (res) {

        }, complete: function (res) {

          //取消加载
          that.setData({
            loading: true
          })
        }
      })
    }
```

新闻列表的原理就是，创建一个 scroll-view，纵向滚动，利用 wx:for="{{detaildata}}"循环来创建每一个 cell，难点只在于每一个 cell 的样式，这里我们展示的 cell 样式包含以下几个字段：新闻标题、0～3 张图片、评论数、时间。cell 的实现核心代码如下所示：

```
    <view class="cellClass">

          <!--标题-->
          <text>{{item.title}}</text>

          <!--图片-->
          <view class="imageClass" hidden="{{!item.has_image}}">
            <image class="imageshow" src="{{item.image_list[0].url}}" mode=
"scaleToFill"> </image>
            <image class="imageshow" src="{{item.image_list[1].url}}"></image>
            <image class="imageshow" src="{{item.image_list[2].url}}"></image>
          </view>

          <!--评论时间栏目-->
          <view class="cellbottomClass">
            <text style="margin-right:10rpx">{{item.comment_count}}评论</text>
            <text>{{item.publish_time}}</text>
          </view>
        </view>
```

7.1.4 首页完整代码

- homePage.json：设置标题、导航栏。代码如下所示：

```
    {
       "navigationBarBackgroundColor":"#ffa589",
       "navigationBarTextStyle":"white",
       "navigationBarTitleText":"首页",
       "backgroundColor":"#eeeeee",
       "backgroundTextStyle":"light"
    }
```

- homePage.js：实现网络请求，获取到内容之后修改数组，页面会显示数组中的内容。代码如下所示：

```
    // pages/homePage/homePage.js
```

非卖品，仅供非商业用途或交流学习使用

```
// 总接口
let ApiUrl = "http://wx.leadingdo.com/demo/"
// 新闻类型接口
let tyepUrl = ApiUrl + "news/type.aspx"
// 新闻列表接口
let newsUrl = ApiUrl + "news/list.aspx"

Page({
  data: {
    tArray: [],//新闻类型数组
    loading: true,//是否显示加载
    ishidden: true,
    curpage: 0,//新闻列表坐标
    listpage: 0, //列表当前页码
    detaildata: [],//新闻列表
    category: "all",//当前分类
    viewHeight:500 //scroll-view 高度
  },
  onReady: function () {

    //创建动画实例
    this.animation = wx.createAnimation({

      //动画持续时间
      duration: 2000,

      //  *  linear  动画一直较为均匀
      //  *  ease    从匀速到加速再到匀速
      //  *  ease-in 缓慢到匀速
      //  *  ease-in-out 从缓慢到匀速再到缓慢
      timingFunction: "ease",
    })

    //读取屏幕高度
    var res = wx.getSystemInfoSync();
    var width = res.screenHeight - 40 - 50;

    //设置 scroll-view 高度
    this.setData({
      viewHeight: width
    });
  },
  // 页面加载
  onLoad: function () {

    //导航栏显示加载状态
    wx.showNavigationBarLoading();

    // 定义 this 代理，处理网络返回数据，不能直接使用 this
    var that = this;

    //请求网络，获取 type
    wx.request({
      url: tyepUrl,
      data: {
      },
      header: {
```

非卖品，仅供非商业用途或交流学习使用

```
        'content-type': 'application/json'
      },
      success: function (res) {

        //获取返回的数组
        let dataArr = [];
        dataArr = res.data.data.data;

        //打印输出
        console.log(dataArr);

        //变量赋值
        that.setData({
          tArray: dataArr
        });
      }, fail: function (res) {

      }, complete: function (res) {

        //取消导航栏加载
        wx.hideNavigationBarLoading();

      }
    })

    //初始化页码从 0 开始
    this.setData({
      listpage: 0
    });

    //默认显示所有新闻
    this.readList("all")
  },
  // 类型单击事件
  typeClick(e) {

    var idx = e.currentTarget.dataset.idx;
    console.log(idx);

    var that = this;
    that.setData({
      curpage: e.target.id
    });

    //初始化页码从 0 开始
    this.setData({
      listpage: 0
    });

    console.log("curpage=", this.data.curpage);
    console.log("listpage==", this.data.listpage);

    //设置分类
    this.setData({
      category: idx
    });
```

非卖品，仅供非商业用途或交流学习使用

```
    //获取新闻
    this.readList()
  },
  //加载更多 scroll-view bindscrolltolower 事件
  addMoreData(e) {
    //页码+1，继续获取新闻
    var that = this;
    var pageTemp = (this.data.listpage + 1)
    that.setData({
      listpage: pageTemp
    });
    //获取新闻
    this.readList()
  },
  //获取新闻列表
  readList() {
    // // 显示加载状态
    this.setData({
      loading: false
    })
    //请求网络，获取 type
    var that = this;
    wx.request({
      url: newsUrl,
      method: "POST",//默认 GET
      data: {
        "category": this.data.category,
        "page": this.data.curpage
      },
      header: {
        //以表单形式提交
        "content-type": "application/x-www-form-urlencoded"

        //以 json 形式提交
        //  "content-type": "application/json
      },
      success: function (res) {

        //如果是第 1 页，坐标是 0，数组先清空
        if (that.data.listpage == 0) {
          that.setData({
            detaildata: [],
          });
        }
        //解析数据
        var arr = res.data.data;
        console.log(arr)
        var dataArr = [];
        for (var index of arr) {
          //转换 Json 字符串为 Json 对象
          dataArr.push(JSON.parse(index.content));
        }
        //修改数组
        that.setData({
          detaildata: dataArr,
        });
      }, fail: function (res) {
```

非卖品，仅供非商业用途或交流学习使用

```
        }, complete: function (res) {
          //取消加载
          that.setData({
            loading: true
          })
        }
      })
    }
  })
```

- homePage.wxml：实现页面的控件布局。代码如下所示：

```
<!--pages/homePage/homePage.wxml-->
<view class="mainViewClass">

  <!--加载动画-->
  <loading hidden="{{loading}}">
    加载中...
  </loading>

  <!--顶部 分类栏目-->
  <view class="topTabBarClass">

    <!--类型 滚动视图-->
    <scroll-view scroll-x="true" style="width: 90%;height: 100%; white-
space: nowrap;display: flex">
      <block wx:for="{{tArray}}">
        <view animation="{{animation}}" class="topTypeListClass {{index==
curpage? 'curPage':''}}" bindtap="typeClick" id="{{index}}" data-idx="{{item.
category}}">
          {{item.name}}
        </view>
      </block>
    </scroll-view>

    <!--右边按钮-->
    <view class="rightview">
      <image mode="aspectFit" src="../../imges/addChannel.png" class=
"addImage"> </image>
    </view>
  </view>

  <!--新闻列表-->
  <!--scroll-y="true":垂直滚动-->
  <!--高度: viewHeight-->
  <!--lower-threshold: 距离底部 50，调用 addMoreData 事件-->
  <!--下拉到底部触发 addData 事件，加载更多-->
  <scroll-view scroll-y="true" style="width: 100%;height:{{viewHeight}}px"
lower-threshold="50" bindscrolltolower="addMoreData">

    <!--循环添加每一个 cell 内容-->
    <block wx:for="{{detaildata}}">
      <view class="cellClass">

        <!--标题-->
        <text>{{item.title}}</text>
```

非卖品，仅供非商业用途或交流学习使用

```
          <!--图片-->
          <view class="imageClass" hidden="{{!item.has_image}}">
            <image  class="imageshow"  src="{{item.image_list[0].url}}"  mode=
"scaleToFill"></image>
            <image class="imageshow" src="{{item.image_list[1].url}}"></image>
            <image class="imageshow" src="{{item.image_list[2].url}}"></image>
          </view>

          <!--评论时间栏目-->
          <view class="cellbottomClass">
            <text style="margin-right:10rpx">{{item.comment_count}}评论</text>
            <text>{{item.publish_time}}</text>
          </view>
        </view>
      </block>
    </scroll-view>
  </view>
```

● homePage.wxss：CSS 样式，主要用来设置页面控件的相关属性（字体、颜色、布局效果等）。代码如下所示：

```
/* pages/homePage/homePage.wxss */

/*主视图*/
.mainViewClass {
  width: 100%;
}

/*顶部分类栏目*/
.topTabBarClass {
  background-color: #ffa589;
  display: flex;
  flex-direction: row;
  width: 100%;
  height: 80rpx;
}

/*类型选择 默认样式*/
.topTypeListClass {
  display: inline-block;
  margin-top: 12rpx;
  margin-left: 10rpx;
  font-size: 30rpx;
  width: 100rpx;
  height: 100%;
  text-align: center;
  z-index: 99;
  color: #333;
}

/*类型选择 选中样式*/
.curPage {
  animation-fill-mode: forwards;
  color: #fff;
  font-size: 35rpx;
```

非卖品，仅供非商业用途或交流学习使用

```
}

/*右侧添加按钮背景视图样式*/
.rightview {
  background-color:transparent;
  height: 100%;
  width: 10%;
  position: relative;
}

/*右侧添加按钮样式*/
.addImage {
    position: absolute;
    width: 40rpx;
    height: 40rpx;
    left: 0;
    top: 0;
    right: 0;
    bottom: 0;
    margin: auto;
}

/*cell 样式*/
.cellClass {
    width: 100%;
    box-sizing:  border-box;
    padding:20rpx;
    background: white;
    border-bottom: 1rpx solid darkgray;
}

/*评论时间栏目 样式*/
.cellbottomClass {
    width: 100%;
    height: 44rpx;
    background-color: transparent;
    color: #dddddd;
    font-size: 30rpx;
    margin-top: 10rpx;
    margin-left: 10rpx;
}

/*图片容器视图 样式*/
.imageClass {
    width: 100%;
    flex-direction: row;
    display: flex;
    height: 150rpx;
    background-color: transparent;
    justify-content: space-around;
}

/*图片样式*/
.imageshow {
    width: 230rpx;
    height: 150rpx;
}
```

非卖品，仅供非商业用途或交流学习使用

7.1.5 用户中心界面实现

用户中心界面显示效果如图 7-3 所示。

图 7-3 用户中心界面效果

1）顶部登录模块实现

开发者可以通过 CSS 样式来设置容器 view 的背景图，在 view 上面添加两个 view，第 1 个 view 添加微信、QQ、微博、电话登录按钮，第 2 个 view 添加“更多登录方式”按钮。代码如下所示：

```
      <!--顶部视图  各登录入口-->
      <view class="topviewClass">
        <view class="topIconClass">
          <image class="images" src="../../imges/UserCenter/wxIcon.png" mode=
"aspectFit" style="width: 60px; height: 60px; background-color: transparent;">
</image>
          <image class="images" src="../../imges/UserCenter/qqIcon.png" mode=
"aspectFit" style="width: 60px; height: 60px; background-color: transparent;">
</image>
          <image class="images" src="../../imges/UserCenter/sinaIcon.png" mode=
"aspectFit" style="width: 60px; height: 60px; background-color: transparent;">
</image>
          <image class="images" src="../../imges/UserCenter/phoneIcon.png" mode=
"aspectFit" style="width: 60px; height: 60px; background-color: transparent;">
</image>
        </view>
        <button class="morebutton">更多登录方式 ></button>
      </view>
```

2）收藏、夜间、设置、历史栏目实现

在横向 view 基础上面添加 4 个 view，每一个 view 上面添加一个 image 图片和 text 文

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

本组件，代码如下所示：

```
        <!--收藏、夜间、设置、历史栏目-->
         <view class="cardBackClass">
           <view class="cardClass">
             <image  src="../../imges/UserCenter/favoriteIcon.png"  mode=
"aspectFit" style="width: 30px; height: 30px; background-color: transparent;">
</image>
             <text> 收藏 </text>
           </view>
           <view class="cardClass">
             <image  src="../../imges/UserCenter/nightIcon.png" mode="aspectFit"
style="width: 30px; height: 30px; background-color: transparent;"></image>
             <text> 夜间 </text>
           </view>
           <view class="cardClass">
             <image src="../../imges/UserCenter/setIcon.png" mode="aspectFit"
style="width: 30px; height: 30px; background-color: transparent;"></image>
             <text> 设置 </text>
           </view>
           <view class="cardClass">
             <image src="../../imges/UserCenter/historyIcon.png" mode="aspectFit"
style="width: 30px; height: 30px; background-color: transparent;"></image>
             <text> 历史 </text>
           </view>
         </view>
```

3）cell 实现

本界面实现了 3 组 cell 的展示，根据不同数组来显示具体多少行，cell 本身引用 template 模板，代码如下所示：

```
    <!--定义 cell 模板-->
    <template name="cellTemplate">
      <text style="text-align:left"> {{item.imagename}} </text>
      <image src="{{item.imagesrc}}" mode="aspectFit" style="width: 20px; height:
20px; background-color: transparent;"></image>
    </template>

    <!--第 1 组 cell-->
         <view class="sectionClass">
        <view class="cellClass">
          <template is="cellTemplate" data="{{item:section1}}" />
        </view>
        </view>

        <!--第 2 组 cell-->
         <view class="sectionClass">
        <block wx:for="{{section2}}" wx:for-item="section2">
          <view class="cellClass">
            <template is="cellTemplate" data="{{item:section2}}" />
          </view>
        </block>
          </view>

        <!--第 3 组 cell-->
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    <view class="sectionClass">
    <block  wx:for="{{section3}}" wx:for-item="section3">
      <view class="cellClass">
        <template is="cellTemplate" data="{{item:section3}}" />
      </view>
    </block>
      </view>
```

7.1.6 用户中心界面完整代码

1）userCenter.json：设置标题和导航栏

```
{
    "navigationBarBackgroundColor":"#ffa589",
    "navigationBarTextStyle":"white",
    "navigationBarTitleText":"个人中心",
    "backgroundColor":"#eeeeee",
    "backgroundTextStyle":"light"
}
```

2）userCenter.js：设置数据

```
// pages/userCenter/userCenter.js
Page({
  data: {

    section1: {
      imagesrc: "../../imges/UserCenter/arrowRight.png",
      imagename: "我的动态"
    },
    section2: [{
      imagesrc: "../../imges/UserCenter/arrowRight.png",
      imagename: "摇一摇"
    }, {
      imagesrc: "../../imges/UserCenter/arrowRight.png",
      imagename: "附近的人"
    },
    ],
    section3: [{
      imagesrc: "../../imges/UserCenter/arrowRight.png",
      imagename: "我的消息"
    }, {
      imagesrc: "../../imges/UserCenter/arrowRight.png",
      imagename: "我的钱包"
    }, {
      imagesrc: "../../imges/UserCenter/arrowRight.png",
      imagename: "我的任务"
    }, {
      imagesrc: "../../imges/UserCenter/arrowRight.png",
      imagename: "意见反馈"
    }, {
      imagesrc: "../../imges/UserCenter/arrowRight.png",
      imagename: "关于我们"
    }]
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    },
    onLoad: function (options) {
      // 页面初始化 options 为页面跳转所带来的参数
    },
    onReady: function () {
      // 页面渲染完成
    },
    onShow: function () {
      // 页面显示
    },
    onHide: function () {
      // 页面隐藏
    },
    onUnload: function () {
      // 页面关闭
    }
  })
```

3）userCenter.wxml：组件布局

```
<!--pages/userCenter/userCenter.wxml-->

<!--定义 cell 模板-->
<template name="cellTemplate">
  <text style="text-align:left"> {{item.imagename}} </text>
  <image src="{{item.imagesrc}}" mode="aspectFit" style="width: 20px;
height: 20px; background-color: transparent;"></image>
</template>

<!--主视图-->
<view class="mainViewClass">

  <!--顶部视图  各登录入口-->
  <view class="topviewClass">
    <view class="topIconClass">
      <image class="images" src="../../imges/UserCenter/wxIcon.png" mode=
"aspectFit" style="width: 60px; height: 60px; background-color: transparent;">
</image>
      <image class="images" src="../../imges/UserCenter/qqIcon.png" mode=
"aspectFit" style="width: 60px; height: 60px; background-color: transparent;">
</image>
      <image class="images" src="../../imges/UserCenter/sinaIcon.png" mode=
"aspectFit" style="width: 60px; height: 60px; background-color: transparent;">
</image>
      <image class="images" src="../../imges/UserCenter/phoneIcon.png" mode=
"aspectFit" style="width: 60px; height: 60px; background-color: transparent;">
</image>
    </view>
    <button class="morebutton">更多登录方式 ></button>
  </view>


  <view class="bottomview">
  <!--定义滚动视图-->
    <scroll-view scroll-y="true">

      <!--收藏、夜间、设置、历史、cell 栏目-->
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        <view class="cardBackClass">
          <view class="cardClass">
            <image src="../../imges/UserCenter/favoriteIcon.png" mode= "aspectFit"
style="width: 30px; height: 30px; background-color: transparent;"></image>
            <text> 收藏 </text>
          </view>
          <view class="cardClass">
            <image  src="../../imges/UserCenter/nightIcon.png"  mode="aspectFit"
style="width: 30px; height: 30px; background-color: transparent;"></image>
            <text> 夜间 </text>
          </view>
          <view class="cardClass">
            <image src="../../imges/UserCenter/setIcon.png" mode="aspectFit"
style="width: 30px; height: 30px; background-color: transparent;"></image>
            <text> 设置 </text>
          </view>
          <view class="cardClass">
            <image src="../../imges/UserCenter/historyIcon.png" mode="aspectFit"
style="width: 30px; height: 30px; background-color: transparent;"></image>
            <text> 历史 </text>
          </view>
        </view>

        <!--第 1 组 cell-->
         <view class="sectionClass">
        <view class="cellClass">
          <template is="cellTemplate" data="{{item:section1}}" />
        </view>
        </view>

        <!--第 2 组 cell-->
         <view class="sectionClass">
        <block wx:for="{{section2}}" wx:for-item="section2">
          <view class="cellClass">
            <template is="cellTemplate" data="{{item:section2}}" />
          </view>
        </block>
          </view>

        <!--第 3 组 cell-->
         <view class="sectionClass">
        <block  wx:for="{{section3}}" wx:for-item="section3">
          <view class="cellClass">
            <template is="cellTemplate" data="{{item:section3}}" />
          </view>
        </block>
          </view>

      </scroll-view>
    </view>
  </view>
```

4）userCenter.wxss：CSS 组件的样式

```
/* pages/userCenter/userCenter.wxss */
.mainViewClass {
    height: 1134rpx;
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    width: 100%;
}

/*顶部视图背景样式*/
.topviewClass {
    background:url(../../imges/UserCenter/TopBackImage.jpg);
    background-size: cover;
    width: 100%;
    height: 30%;
}

/*顶部各按钮样式(微信、QQ、微博、手机登录)*/
.topIconClass {
    width: 90%;
    height: 80%;
    background-color: transparent;
    margin-left: 5%;
    display: flex;
    flex-direction: row;
    justify-content: space-around;
    align-items: center;
}


/*收藏、夜间、设置、历史、cell 栏目*/
.bottomview{
    background-color: lightgray;
    height: 70%;
    width: 100%;
}

/*更多按钮*/
.morebutton{
    margin-top: -40rpx;
    font-size: 14px;
    color: white;
    background-color: #999999;
    width: 250rpx;
    height: 66rpx;
    border-radius: 100rpx;
}

/*收藏、夜间、设置、历史 底部视图样式*/
.cardBackClass{
    width: 100%;
    height: 120rpx;
    display: flex;
    flex-direction: row;
    justify-content: space-around;
    align-items: center;
    background-color: white;
}

/*收藏、夜间、设置、历史 按钮样式*/
.cardClass {
    display: flex;
    flex-direction: column;
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    font-size: 28rpx;
}

/*组样式*/
.sectionClass {
    margin-top: 20rpx;
}

/*cell 样式*/
.cellClass {
    background-color: white;
    width: 100%;
    height: 90rpx;
    display: flex;
    flex-direction: row;
    justify-content:space-between;
    align-items: center;
    border-bottom: 1rpx solid lightgray;
    box-sizing: border-box;
    padding-left: 16rpx;
    padding-right: 16rpx;
}
```

7.2 书架功能

本节主要介绍几种书架的布局，使用本地图片作为演示，大家在开发项目时，可以使用网络图片，以减少小程序包的大小。书架显示效果如图 7-4 所示。

图 7-4 书架小程序

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

7.2.1 精彩推荐模块实现

使用 swiper 组件实现多图轮播，具体代码如下所示：

```
    <!--精彩推荐 栏目-->
      <view class="topBarClass">
        <image src="../../images/icon.png" class="icon"></image>
        <text>  精彩推荐</text>
        <image src="../../images/mark.png" class="book-icon"></image>
      </view>

      <!--顶部滚动视图-->
      <swiper indicator-dots="true" autoplay="true" interval="6000" duration=
"3000">
        <block wx:for="{{recommendBooks}}">
          <swiper-item>
            <image src="{{item.image}}" class="slide-image" data-purchase=
"{{item.url}}" bindtap="bindToDetailTap"/>
          </swiper-item>
        </block>
      </swiper>
```

block 中利用 wx:for 循环添加 recommendBooks 数组中的内容，在每一个 swiper-item 中添加 image 组件，bindtap="bindToDetailTap"用来进行事件绑定，data-purchase="{{item.url}}"用来设置单击事件的参数。单击事件代码如下所示：

```
    bindToDetailTap: function (e) {
       wx.navigateTo({
         url: '../detail/detail?url=' + e.currentTarget.dataset.url
       })
     }
```

小程序通过 e.currentTarget.dataset.url 来获取 data-purchase="{{item.url}}"中的 item.url 值。

7.2.2 热门书籍模块实现

在 view 中利用 wx:for 循环添加 hotBooks 数组中的内容，每一个 view 中包含 image 图片组件和 text 文本组件，具体代码如下所示：

```
    <!--热门书单 标题栏目-->
      <view class="hot-books">
        <view class="hb-bar">
          <text> 热门书籍</text>
          <image src="../../images/mark.png" class="book-icon"></image>
        </view>

    <!--热门书籍 板块-->
        <view class="hb-booklist">
          <view class="hb-book" wx:for="{{hotBooks}}" bindtap="bindToDetailTap"
data-purchase="{{item.url}}" data-id="{{item.id}}">
            <image src="{{item.image}}"></image>
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
          <text class="book-name">{{item.name}}</text>
        </view>
      </view>
    </view>
```

上面的代码使用 bindtap 绑定 bindToDetailTap 单击事件。

7.2.3 精品书籍模块实现

利用 wx:for 添加 view，每一个 view 上面有 2 个 image 图片组件（广告和作者头像）和 2 个 text 组件（书名和简介）。具体代码如下所示：

```
  <!--精品书籍 标题模块-->
    <view class="boutique-books">
      <view class="bb-bar">
        <text> 精品书籍</text>
        <image src="../../images/mark.png" class="book-icon"></image>
      </view>

      <!--精品书籍列表-->
      <view class="bb-booklist">
        <view class="book" wx:for="{{boutiqueBooks}}" data-id="{{item.id}}"
data-purchase="{{item.url}}" bindtap="bindToDetailTap">
          <image src="{{item.images.title_img}}" class="title-img"></image>
          <image src="{{item.images.author}}" class="author"></image>
          <text class="book-name">{{item.name}}</text>
          <text class="book-summary">{{item.miniSummary}}</text>
        </view>
      </view>
    </view>
```

关于本节书架小程序的几个重要文件，下面详细逐一介绍。

- app.json 文件：实现整个小程序的页面结构。完整代码如下所示：

```
{
  "pages": [
    "pages/home/home",
    "pages/myBooks/myBooks",
    "pages/search/search",
    "pages/detail/detail"
  ],
  "window": {
    "backgroundTextStyle": "dark",
    "navigationBarBackgroundColor": "#FFA589",
    "navigationBarTitleText": "首页",
    "navigationBarTextStyle": "white",
    "backgroundColor": "#EEEEEE",
    "enablePullDownRefresh":"true"
  },
  "tabBar": {
    "list": [
      {
        "pagePath": "pages/home/home",
        "text": "朗读广场",
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
            "iconPath": "images/TabbarHomeN.png",
            "selectedIconPath": "images/TabbarHomeS.png"
          },
          {
            "pagePath": "pages/search/search",
            "text": "寻找朗读",
            "iconPath": "images/TabbarSearchN.png",
            "selectedIconPath": "images/TabbarSearchS.png"
          },{
            "pagePath": "pages/myBooks/myBooks",
            "text": "我的朗读",
            "iconPath": "images/TabbarCollectN.png",
            "selectedIconPath": "images/TabbarCollectS.png"
          }
        ],
        "borderStyle": "white",
        "backgroundColor": "#f0f0f0",
        "selectedColor": "#1aad16"
      }
    }
```

- home.json：首页导航栏配置信息。代码如下所示：

```
    {
        "navigationBarBackgroundColor":"#ffa589",
        "navigationBarTextStyle":"white",
        "navigationBarTitleText":"朗读广场",
        "backgroundColor":"#eeeeee",
        "backgroundTextStyle":"light"
    }
```

- home.wxml：首页页面布局，代码如下所示：

```
    <!--pages/home/home.wxml-->
    <view class="mainClass">

    <!--精彩推荐 栏目-->
      <view class="topBarClass">
        <image src="../../images/icon.png" class="icon"></image>
        <text>   精彩推荐</text>
        <image src="../../images/mark.png" class="book-icon"></image>
      </view>

      <!--顶部滚动视图-->
      <swiper indicator-dots="true" autoplay="true" interval="6000" duration=
"3000">
        <block wx:for="{{recommendBooks}}">
          <swiper-item>
            <image src="{{item.image}}" class="slide-image" data-purchase=
"{{item.url}}" bindtap="bindToDetailTap"/>
          </swiper-item>
        </block>
      </swiper>

    <!--热门书单 标题栏目-->
      <view class="hot-books">
        <view class="hb-bar">
```

非卖品，仅供非商业用途或交流学习使用

```
        <text> 热门书籍</text>
        <image src="../../images/mark.png" class="book-icon"></image>
      </view>

    <!--热门书籍板块-->
      <view class="hb-booklist">
        <view class="hb-book" wx:for="{{hotBooks}}" bindtap="bindToDetailTap"
data-purchase="{{item.url}}" data-id="{{item.id}}">
          <image src="{{item.image}}"></image>
          <text class="book-name">{{item.name}}</text>
        </view>
      </view>
    </view>

    <!--精品书籍 标题模块-->
    <view class="boutique-books">
      <view class="bb-bar">
        <text> 精品书籍</text>
        <image src="../../images/mark.png" class="book-icon"></image>
      </view>

      <!--精品书籍列表-->
      <view class="bb-booklist">
        <view class="book" wx:for="{{boutiqueBooks}}" data-id="{{item.id}}"
data-purchase="{{item.url}}" bindtap="bindToDetailTap">
          <image src="{{item.images.title_img}}" class="title-img"></image>
          <image src="{{item.images.author}}" class="author"></image>
          <text class="book-name">{{item.name}}</text>
          <text class="book-summary">{{item.miniSummary}}</text>
        </view>
      </view>
    </view>
  </view>
```

- home.wxss：首页页面 CSS 样式，代码如下所示：

```
/* pages/home/home.wxss */

/*页面样式*/
page{
  width: 100%;
  height: 100%;
}

/*主视图样式*/
.mainClass{
  width: 100%;
  height: 100%;
  font-family: "Microsoft YaHei"
}

/*顶部 推荐模块 滚动视图样式*/
.topBarClass{
  height: 60rpx;
  padding: 0 6rpx;
  margin:10rpx 0;
  border-left:10rpx solid #ffa589;
```

非卖品，仅供非商业用途或交流学习使用

```
  display: flex;
  align-items: flex-end;
}
.topBarClass .icon{
  width: 176rpx;
  height: 60rpx;
}
.topBarClass .book-icon{
  width: 40rpx;
  height: 40rpx;
}
.topBarClass text{
  font: 30rpx "SimSun";
  color:#494949;
  font-weight: bold;
}

/*滚动图片区域样式*/
swiper{
    width: 750rpx;
  height: 340rpx;
}
.slide-image{
  width: 750rpx;
  height: 340rpx;
}

/*热门书籍*/
.hot-books{
  padding: 20rpx 0;
  height: 440rpx;
  margin-bottom: 10rpx;
  box-shadow: 0 2rpx 4rpx 4rpx #ededed;
}
.hb-bar{
  height: 60rpx;
  padding: 0 20rpx;
  margin:10rpx 0 20rpx 0;
  border-left:10rpx solid #ffa589;
  display: flex;
  align-items: flex-end;
}
.hb-bar text{
  font: 30rpx "SimSun";
  color:#494949;
  font-weight: bold;
}
.hb-bar image{
  width: 40rpx;
  height: 40rpx;
}
.hb-booklist{
  display: flex;
   justify-content: space-around;
}
.hb-book{
 display: flex;
```

非卖品，仅供非商业用途或交流学习使用

```
  flex-direction: column;
  align-items: flex-start;
}
.hb-book image{
  width: 170rpx;
  height: 251rpx;
}
.hb-book .book-name{
  display: block;
  max-width: 170rpx;
  overflow: hidden;
  white-space: nowrap;
  text-overflow: ellipsis;
  font-size: 26rpx;
  font-weight: bold;
  color:#494949;
  padding: 16rpx 0;
}
.hb-book progress{
  width: 120rpx;
}
.book-grade{
  display: flex;
}
.book-grade .grade{
  padding-left: 20rpx;
  font-size: 24rpx;
  color: #FDB235;
}
.boutique-books{
  padding: 20rpx 0 40rpx 0;
  height: 440rpx;
}
.bb-bar{
  height: 60rpx;
  padding: 0 20rpx;
  margin:10rpx 0;
  border-left:10rpx solid #ffa589;
  display: flex;
  align-items: flex-end;
}
.bb-bar text{
  font: 30rpx "SimSun";
  color:#494949;
  font-weight: bold;
}
.bb-bar image{
  width: 40rpx;
  height: 40rpx;
}

.bb-booklist .book{
 margin-bottom: 30rpx;
 position: relative;
 display: flex;
 flex-direction: column;
}
```

非卖品，仅供非商业用途或交流学习使用

```
.bb-booklist .book .title-img{
  width: 750rpx;
  height: 250rpx
}
.bb-booklist .book .author{
  position: absolute;
  top:220rpx;
  right: 50rpx;
  width: 60rpx;
  height: 60rpx;
  border: 2rpx solid #fff;
  border-radius: 50%;
}
.bb-booklist .book .book-name{
  font-size: 34rpx;
  color:#494949;
  font-weight: bold;
  padding: 20rpx 30rpx;
}
.bb-booklist .book .book-summary{
  padding: 0 30rpx;
  font-size: 26rpx;
  color:#8b8b8b;
}
```

- home.js：首页 js 脚本，代码如下所示：

```
// pages/home/home.js
//获取应用实例
var app = getApp()
Page({
  data: {
    recommendBooks: [
      {
        name: "Swift 精讲 2",
        image: "../../images/book0.jpg",
        url: "https://item.jd.com/11937350.html"
      },
      {
        name: "Swift 精讲 1",
        image: "../../images/book1.jpg",
        url: "https://item.jd.com/11672854.html"
      },
      {
        name: "Swift 入门",
        image: "../../images/book2.jpg",
        url: "https://item.jd.com/11706321.html#crumb-wrap"
      }],
    hotBooks: [
      {
        name: "Swift 精讲 2",
        image: "../../images/hotBook0.jpg",
        url: "https://item.jd.com/11937350.html"
      },
      {
        name: "Swift 精讲 1",
        image: "../../images/hotBook1.jpg",
```

非卖品，仅供非商业用途或交流学习使用

```
        url: "https://item.jd.com/11672854.html"
      },
      {
        name: "Swift 入门",
        image: "../../images/hotBook2.jpg",
        url: "https://item.jd.com/11706321.html#crumb-wrap"
      }],
    boutiqueBooks: [
      {
        name: "Swift 精讲 2",
        miniSummary:"※本书的内容迭代更新至 Swift2.0 版本，增加了 60 余页干货，主要讲解如何使用 Swift 语言开发 App，是初学者从零起步学习 App 开发的佳选。",
        images: {
          title_img: "../../images/boutiquebook0.jpg",
          author: "../../images/head.jpg"
        },
        url: "https://item.jd.com/11937350.html"
      },
      {
        name: "Swift 精讲 1",
        miniSummary:"※在精细讲解基础知识之后，本书通过实践项目来讲解应用方法，帮助读者快速掌握知识。其中的图片素材和源代码可供下载，使你可以快速上手。",
        images: {
          title_img: "../../images/boutiquebook1.jpg",
          author: "../../images/head.jpg"
        },
        url: "https://item.jd.com/11937350.html"
      },
      {
        name: "Swift 入门",
        miniSummary:"“爱上 Swift”系列丛书荣登 2015 年的年度好书排行榜，以其专注实战、学以致用的特色，成为 Swift 门类中受到读者喜爱的丛书。",
        images: {
          title_img: "../../images/boutiquebook2.jpg",
          author: "../../images/head.jpg"
        },
        url: "https://item.jd.com/11937350.html"
      }]
  },
  onLoad: function () {
  },
  bindToDetailTap: function (e) {
    wx.navigateTo({
      url: '../detail/detail?url=' + e.currentTarget.dataset.url
    })
  }
})
```

7.3 录音功能

本节主要介绍录音功能的实现，包括录音之后保存文件，界面显示所有录音列表，单击可以播放录音。显示效果如图 7-5 所示。

非卖品，仅供非商业用途或交流学习使用

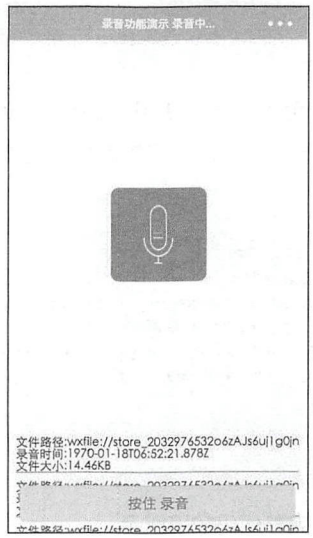


图 7-5 录音功能

首先创建一个按钮，添加按钮的 bindtouchstart 按下事件和 bindtouchend 释放事件。代码如下所示：

```
    <!--录音按钮-->
     <view class="recordClass">
     <button class="recordButtonClass" bindtouchstart="touchDown" bindtouchend=
"touchUp">按住 录音</button>
     </view>
```

录音动画的显示功能，主要是用了 5 张图片，根据计时器的数字显示对应 picId 的图片，具体代码如下所示：

```
    <!--录音音量动画显示图片-->
    <!--开始录音，显示该 view-->
    <view  wx:if="{{isSpeaking}}"  class="speakPicViewClass">
    <!--音量图片 1-->
    <image wx:if="{{picId==1}}" class="imageClass" src="../../images/sound1.
png" ></image>

    <!--音量图片 2-->
    <image wx:if="{{picId==2}}" class="imageClass" src="../../images/sound2.
png" ></image>

    <!--音量图片 3-->
    <image wx:if="{{picId==3}}" class="imageClass" src="../../images/sound3.
png" ></image>

    <!--音量图片 4-->
    <image wx:if="{{picId==4}}" class="imageClass" src="../../images/sound4.
png" ></image>

    <!--音量图片 5-->
```

非卖品，仅供非商业用途或交流学习使用

```
    <image  wx:if="{{picId==5}}"class="imageClass"  src="../../images/sound5.
png" ></image>
     </view>
```

列表的显示相对简单，最常用的功能是使用 scroll-view 通过 wx:for="{{voices}}创建 view，每一个 view 上面有 3 行 view，显示对应的内容。具体代码如下所示：

```
<!--显示录音的列表-->
<scroll-view>
 <block  wx:for="{{voices}}">
    <view class="cellClass">
   <view class="cellRowClass" data-key="{{item.filePath}}" bindtap="playAudio" >
      <view  class="dateClass">文件路径:{{item.filePath}}</view>
    <view  class="dateClass" >录音时间:{{item.createTime}}</view>
    <view  class="dateClass">文件大小:{{item.size}}KB</view>
  </view>
    </view>
</block>
 </scroll-view>
```

CSS 样式文件 index.wxss 完整代码如下所示：

```
/**index.wxss**/

/*音量图片底部 view 样式*/
.speakPicViewClass{
    position: relative;
    height: 240rpx;
    width: 240rpx;
    border-radius: 20rpx;
    margin: 50% auto;
    background: #26A5FF;
}

/*录音按钮所在的 view 样式*/
.recordClass{
    position: fixed;
    bottom: 0;
    left: 0;
    height: 120rpx;
    width: 100%;
}

/*录音按钮 样式*/
.recordButtonClass{
  margin-left: 30rpx;
  margin-right: 30rpx;
}

/*音量图片 样式*/
.imageClass{
  position: absolute;
  width: 74rpx;
  height:150rpx;
  margin-top: 45rpx;
  margin-left: 83rpx;
}
```

非卖品，仅供非商业用途或交流学习使用

```
/*一个 cell 的样式*/
.cellClass {
  overflow: hidden;
  border-bottom: 2rpx solid #26A5FF;
    display: flex;
    margin: 20rpx;
}

.cell .cellRowClass{
    flex:1;
    position: relative;
}

/*一行内容*/
.dateClass{
    font-size: 30rpx;
    text-overflow: ellipsis;
    white-space:nowrap;
    overflow:hidden;
}
```

本节关键的部分也就是 js 中各功能的实现，js 脚本文件的完整代码如下所示：

```
//index.js
//获取应用实例
var app = getApp()
Page({
  data: {
    picId: 1,//图片初始 ID
    isSpeaking: false,//是否开始说话
    voices: [],//音频数组
  },
  onLoad: function () {
    this.readFileList()
  },

  //按钮按下事件
  touchDown: function () {
    console.log("按钮按下-开始录音...")

    var that = this;
    speaking.call(this);
    this.setData({
      isSpeaking: true
    })

    //开始录音
    wx.startRecord({
      success: function (res) {

        //临时路径,下次进入小程序时无法正常使用
        var tempFilePath = res.tempFilePath
        console.log("tempFilePath: " + tempFilePath)

        //持久保存
        wx.saveFile({
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
          tempFilePath: tempFilePath,
          success: function (res) {
            //持久保存路径，本地文件存储的大小限制为 100MB
            var savedFilePath = res.savedFilePath
            console.log("savedFilePath: " + savedFilePath)
          }
        })
        wx.showToast({
          title: '录音成功',
          icon: 'success',
          duration: 1000
        })

        //重新读取列表
        that.readFileList()
      },
      fail: function (res) {
        //录音失败
        wx.showModal({
          title: '提示',
          content: '录音错误!',
          showCancel: false,
          success: function (res) {
            if (res.confirm) {
              console.log('单击确定')
            }
          }
        })
      }
    })
  },

  //抬起手指
  touchUp: function () {
    console.log("释放录音功能...")
    this.setData({
      isSpeaking: false,
    })
    clearInterval(this.timer)
    wx.stopRecord()
  },

  //单击播放录音
  playAudio: function (e) {
    var filePath = e.currentTarget.dataset.key;

    //单击开始播放
    wx.showToast({
      title: '开始播放',
      icon: 'success',
      duration: 1000
    })

    //播放录音文件
    wx.playVoice({
      filePath: filePath,
      success: function () {
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
            wx.showToast({
              title: '播放结束',
              icon: 'success',
              duration: 1000
            })
          }
        })
      },

      //获取音频列表
      readFileList: function (e) {

        var that = this;
            //获取录音音频列表
            wx.getSavedFileList({
              success: function (res) {
                var voices = [];
                for (var i = 0; i < res.fileList.length; i++) {

                  //格式化时间
                  var createTime = new Date(res.fileList[i].createTime)

                  //将音频大小 B 转为 KB
                  var size = (res.fileList[i].size / 1024).toFixed(2);
                  var voice = { filePath: res.fileList[i].filePath, createTime:
createTime, size: size };
                  console.log("文件路径: " + res.fileList[i].filePath)
                  console.log("文件时间: " + createTime)
                  console.log("文件大小: " + size)
                  voices = voices.concat(voice);
                }
                that.setData({
                  voices: voices
                })
              }
            })
      }
    })

    //话筒帧动画
    function speaking() {
      var that = this;
      var i = 1;
      this.timer = setInterval(function () {
        i++;
        i = i % 5;
        console.log("==============:"+i);
        that.setData({
          picId: i
        })
      }, 200);
    }
```

Page 的 data 中有 3 个对象：

1. picId：初始默认值是 1，界面根据 picId 的值来显示不同的图片，实现录音时的动画。

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

2. isSpeaking：初始默认值是 false，表示未开始说话，界面不显示录音动画，此值为 true 时，界面显示录音的动画。

3. voices：音频数组，默认设置为空数组，读取本地所有录音文件，保存在 voices 数组中，界面根据此数组显示列表。

js 脚本文件的完整代码在页面加载方法（onLoad）中，调用了 this.readFileList()事件。readFileList 事件用来读取本地所有文件，读取之后，处理数据，保存到 voices 数组中。voices 数组修改之后，界面自动刷新显示数组内容。

单击“按住录音”按钮，触发 touchDown 事件，执行代码 speaking.call（this），调用 speaking 事件。该方法设置了一个定时期，200ms 修改一次 picId 的值，这样界面会不断显示不同的图片，实现动画效果。.call（this）是指将 speaking 事件中的 timer 添加到 this 上面。录音结束之后需要清除 timer，这样才可以停止计时器。

紧接着，代码调用 API 实现录音功能 wx.startRecord。录音成功之后，永久保存文件，并且重新读取本地音频文件列表。

touchUp 释放按钮事件，主要用于停止录音，清除 time 定时器。

playAudio 列表单击事件，实现播放音频功能。

7.4 二维码生成器

任何项目在开发过程中都会遇到一些自己一时难以实现的功能，而此功能网上会有一些开源的代码或框架已实现得很好，这个时候，我们可以将其拿来使用或研究实现的原理。

本节主要介绍利用一个开源 js 代码，生成二维码图片。显示效果如图 7-6 所示。

在这里我们从网上下载了一个 js 文件 wxqrcode.js，如图 7-7 所示。

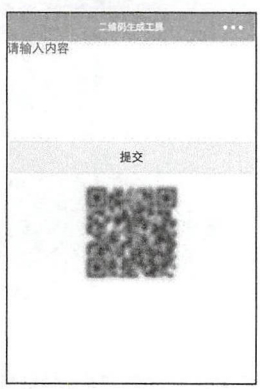


图 7-6 二维码生成器

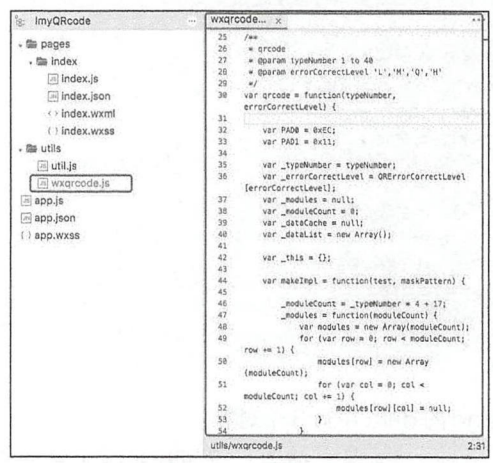


图 7-7 第三方 js 源代码

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

在 index.js 文件中使用 import 引入第三方 js，代码如下所示：

```
import QR from "../../utils/wxqrcode.js" // 二维码生成器
```

index.wxml 界面布局代码如下所示：

```
<!--pages/index/index.wxml-->
<view>
  <form bindsubmit="bindFormSubmit">
    <textarea placeholder="请输入内容" name="textarea"/>
    <button form-type="submit"> 提交 </button>
  </form>
<image src="{{imagePath}}" class="qrImagecodeClass"></image>
</view>
```

该段代码创建 form 表单，textarea 用来输入文本，bindsubmit="bindFormSubmit"绑定表单的“提交”按钮事件。单击按钮，获取文本框内容，然后生成二维码。

index.wxss 文件代码如下所示：

```
/* pages/index/index.wxss */
/*二维码图片样式*/
.qrImagecodeClass {
  width: 280rpx;
  height: 280rpx;
  display: block;
  margin: 0 auto;
  margin-top: 20px;
}
```

index.js 脚本文件完整代码如下所示：

```
// pages/index/index.js
import QR from "../../utils/wxqrcode.js" // 二维码生成器

Page({
    data: {

        // 要生成二维码的文本
        text: 'https://item.jd.com/11937350.html',

        //图片路径
        imagePath: ''
    },
    onLoad: function () {
        var that = this;

        //生成二维码
        let qrcodeSize = that.getQRCodeSize()
        that.createQRCode(that.data.text, qrcodeSize)
    },

    //按钮事件
    bindFormSubmit: function (e) {

        var that = this;
        console.log(e.detail.value.textarea);
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        let qrcodeSize = that.getQRCodeSize()
        that.createQRCode(e.detail.value.textarea, qrcodeSize)
    },

    //获取画布大小，适配不同屏幕大小的 canvas
    getQRCodeSize: function () {
        var size = 0;
        try {
            var res = wx.getSystemInfoSync();
            var scale = 750 / 278; //不同屏幕下 QRCode 的适配比例；设计高是 750，宽是 278
            var width = res.windowWidth / scale;
            size = width;

        } catch (e) {
            console.log("获取设备信息失败" + e);
        }
        return size;
    },

    //绘制二维码图片
    createQRCode: function (text, size) {

        let that = this
        let _img = QR.createQrCodeImg(text, {
            size: parseInt(size)
        })

        //修改图片路径
        that.setData({
            imagePath: _img
        })
    },
})
```

createQRCode 方法就是创建二维码的方法，调用 QR 的 createQrCodeImg 方法，生成二维码图片。

7.5 图片滤镜

本节主要介绍图片的模糊、怀旧、复古、美白、饱和度、亮度、对比度、动态滤镜功能的实现。其中模糊、怀旧、复古、美白 4 个功能通过 CSS 样式 filter 属性来设置，显示效果如图 7-8 所示。饱和度、亮度、对比度通过 style 属性来设置，显示效果如图 7-9 所示。

7.5.1 模糊、怀旧、复古、美白功能的实现

在 picture.wxml 文件中添加 image 组件，组件 class 通过变量来决定使用哪一个 class。

如果 addblur==1，则 class='addblur'；如果 oldEffect==1，则 class='oldeffect'；如果 addretro==1，则 class='addretro'；如果 addBeati==1，则 class='addBeati'，具体代码如下页所示：

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    <image class="{{addblur == 1 ? 'addblur':''}}{{oldEffect == 1 ?'oldeffect':''}}
{{addretro == 1 ?'addretro':''}}{{addBeati == 1 ? 'addBeati':''}}" src="../../
images/pic2.png"></image>
```

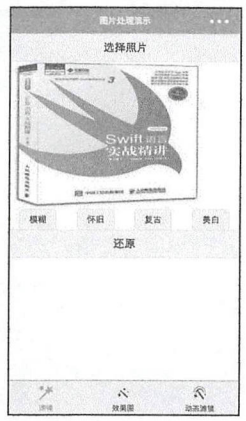


图 7-8 模糊、怀旧、复古、美白

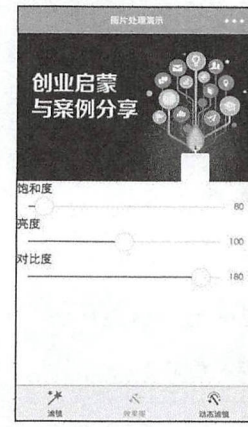


图 7-9 饱和度、亮度、对比度

添加按钮，绑定事件，单击按钮来修改 addblur、oldEffect、addretro、addBeati 的值。具体代码如下所示：

```
    <view class="buttonViewClass">
      <button class="btn" bindtap="blurClick">模糊</button>
      <button class="btn" bindtap="oldClick">怀旧</button>
      <button class="btn" bindtap="retroClick">复古</button>
      <button class="btn" bindtap="beatiClick">美白</button>
    </view>
     <button bindtap="originClick">还原</button>
  </view>

  // 模糊
   blurClick:function(){
     //还原所有数值=0
     this.originClick()

     //设置 addblur=1
     var self = this;
     self.setData({
       addblur:1
     });
   },

   // 怀旧
   oldClick:function(){
     //还原所有数值=0
     this.originClick()

     //设置 oldEffect=1
     var self = this;
     self.setData({
       oldEffect:1
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    });
  },

  // 复古
  retroClick:function(){
    //还原所有数值=0
    this.originClick()

    //设置 addretro=1
    var self = this;
    self.setData({
      addretro:1
    });
  },

  // 美白
  beatiClick:function(){

    //还原所有数值=0
    this.originClick()

    //设置 addBeati=1
     var self = this;
    self.setData({
      addBeati:1
    });
  },

  // 还原
  originClick:function(){
     var self = this;
    self.setData({
      addblur:0,
      oldEffect:0,
      addretro:0,
      addBeati:0
    });
  }

//picture.wxss 文件中 css class 如下所示:
/*模糊*/
.addblur {
   filter: blur(6px);
}

/*怀旧*/
.oldeffect {
   filter: sepia(.5);
}

/*复古*/
.addretro {
   filter: grayscale(1);
}

/*美白*/
.addBeati {
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        filter: brightness(130%);
    }
```

7.5.2 饱和度、亮度、对比度功能的实现

在 picture1.wxml 文件中添加 image 组件，组件通过 style 的 filter 属性来修改图片的饱和度、亮度、对比度。具体代码如下所示：

```
    <image class="imgClass" style="filter:saturate({{saturate}}%) brightness
({{brightness}}%) contrast({{contrast}}%)" src="../../images/pic1.png"></image>
```

saturate 用来设置图片饱和度，brightness 用来设置图片亮度，contrast 用来设置图片对比度。

创建 3 个 slider，分别用来修改 saturate、brightness、contrast 的值。代码如下所示：

```
    <view>饱和度</view>
        <slider bindchange="saturationClick" show-value min="0" max="1000"
value="80" />
        <view>亮度</view>
        <slider bindchange="brightnessClick" show-value min="0" max="200"
value="100" />
        <view>对比度</view>
        <slider bindchange="contrastClick" show-value min="0" max="200"
value="180" />
    </view>
```

picture1.js 脚本文件代码如下所示：

```
    // pages/picture1/picture1.js
    Page({
      data: {
        saturate: 80,
        brightness: 100,
        contrast: 180
      },

      // 饱和度调节
      saturationClick: function (e) {
        var self = this;
        self.setData({
          saturate: e.detail.value
        });
      },

      // 亮度调节
      brightnessClick: function (e) {
        var self = this;
        self.setData({
          brightness: e.detail.value
        });
      },

      //对比度调节
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    contrastClick: function (e) {
      var self = this;
      self.setData({
        contrast: e.detail.value
      });
    }
  })
```

7.5.3 动态滤镜的实现

动态滤镜主要用于执行 CSS 样式里面的关键动画。代码如下所示：

```
<!--pages/picture2/picture2.wxml-->
<view>
    <image class="pic picAnmaion" id="picAnmaion" src="../../images/pic4.
png"> </image>
</view>
```

Image 拥有 picAnmaion 样式。picAnmaion 代码如下所示：

```
/* pages/picture2/picture2.wxss */

/*图片样式*/
.pic {
    margin-top: 20px;
    width: 100%;
}

@keyframes anamiton {
    0% {
        filter: grayscale(.5) blur(1px) saturate(2);
    }
    100% {
        filter: grayscale(.2) blur(6px) saturate(9);
    }
}

/*动画*/
.picAnmaion {
    animation-name: anamiton;
    animation-duration: 4s;
    animation-iteration-count: 10;
    animation-direction: alternate;
    animation-timing-function: ease-in-out;
    animation-fill-mode: forwards;
    animation-delay: 0s;
}
```

picAnmaion 执行@keyframes anamiton 动画。"@keyframes"是出自 CSS3 的 animation 属性，这是一个关键帧动画。0%和100%代表动画的开始状态和结束状态。

animation-name：anamiton；定义动画名称，可以随便起一个。

animation-duration：4s；定义过渡时间，动画持续多少秒。

animation-iteration-count：10；定义动画播放次数。

非卖品，仅供非商业用途或交流学习使用

animation-direction：alternate；定义是否应该轮流反向播放动画。默认值为 normal，取默认值时动画应该正常播放。设置为 alternate，动画应该轮流反向播放。animation-direction 的值是"alternate"，则动画会在奇数次数（1、3、5 等）正常播放，而在偶数次数（2、4、6 等）反向播放。

animation-timing-function：ease-in-out；定义动画的速度曲线。该函数使用名为三次方贝塞尔（Cubic Bezier）函数的数学函数来生成速度曲线。在该函数中能够使用自己的值，也可以使用预定义的值，如表 7-1 所示。

表 7-1　　animation-timing-function 属性值

值	描述
linear	动画从头到尾的速度是相同的
ease	默认。动画以低速开始，然后加快，在结束前变慢
ease-in	动画以低速开始
ease-out	动画以低速结束
ease-in-out	动画以低速开始和结束
cubic-bezier(n,n,n,n)	在 cubic-bezier 函数中自己的值。可能的值是 0～1

animation-fill-mode：forwards；属性规定动画在播放之前或之后，其动画效果是否可见。其属性值是由逗号分隔的一个或多个填充模式关键词。属性值如表 7-2 所示。

表 7-2　　animation-fill-mode 属性值

值	描述
none	不改变默认行为
forwards	动画完成后，保持最后一个属性值（在最后一个关键帧中定义）
backwards	在 animation-delay 所指定的一段时间内，在动画显示之前，应用开始属性值（在第一个关键帧中定义）
both	向前和向后填充模式都被应用

animation-delay：0s；定义动画何时开始。定义动画开始前等待的时间，以 s 或 ms 计。默认值是 0。

7.6　仿电影小应用

电影小应用主要展示正在热映的电影列表、即将上映的电影列表，实现搜索电影功能。显示效果如图 7-10 所示。

正在热映栏目实现了滚动到底部自动加载下一页功能，即将上映栏目实现了到底部需要单击“单击加载更多”按钮，来加载下一页。自动加载功能只不过是 scroll-view 绑定了 bindscrolltolower="scrolltolower"事件。该事件自动调用“单击加载更多”按钮功能。

非卖品，仅供非商业用途或交流学习使用

图 7-10　电影类小应用

7.6.1　电影列表页面的实现

下面将以即将上映为示例，讲解列表功能的实现。

通过 showLoading 值，页面来显示加载动画，showLoading=1 显示加载动画，else 显示列表。代码如下所示：

```
<!--pages/comingSoon/comingSoon.wxml-->

<!--loading 动画效果，根据 showLoading 来显示-->
<block wx:if="{{showLoading}}">
  <view class="loadingViewClass">
    <text class="imageloadingClass" />
    <text class="loadingText">正在加载中</text>
  </view>
</block>

<!--不显示 loading 的话，显示列表-->
<block wx:else>
```

电影滚动列表是在 scroll-view 基础上通过 block wx:for-items="{{films}}"，来添加一个 cell 单元。一个 cell 单元主要包含图片、电影名称、评分、导演、主演，如图 7-11 所示。

图 7-11　cell 单元内容

具体代码如下所示：

```
<!--可以设置 bindscrolltolower="scrolltolower" 上拉加载更多-->
  <scroll-view style="height: 100%;" scroll-y="true" class="container"
bindscroll="scroll" >
```

非卖品，仅供非商业用途或交流学习使用

```
        <!--循环加载列表-->
        <block wx:for-items="{{films}}" wx:for-item="film">
          <view  class="filmItem"  bindtap="viewDetail"  data-id="{{film.id}}"
data-title="{{film.title}}">

            <!--图片-->
            <view class="filmImage">
              <image src="{{film.images.medium}}"></image>
            </view>

            <!--电影名称和上映时间 -->
            <view class="film-info">
              <view>
                <text class="film-title">{{film.title}}</text>
                <text class="film-year">{{film.year}}</text>
              </view>

              <!-- 有评分的效果 -->
              <view class="film-rating">
                <block wx:if="{{film.rating.average > 0}}">
                  <text class="label">评分</text>
                  <text class="rating">{{film.rating.average}}</text>
                </block>

                <!-- 没有评分的效果 -->
                <block wx:else>
                  <text class="label">暂无评分</text>
                </block>
              </view>

              <!--导演-->
              <view class="directors">
                <text class="label">导演</text>
                <block wx:for-items="{{film.directors}}" wx:for-item="director">
                  <text class="person">{{director.name}}</text>
                </block>
              </view>

              <!--主演-->
              <view class="casts">
                <text class="label">主演</text>
                <block wx:for-items="{{film.casts}}" wx:for-item="cast">
                  <text class="person">{{cast.name}}</text>
                </block>
              </view>
            </view>
          </view>
        </block>

        <!--显示加载更多、没有更多-->
        <view class="load-more-wrap">
          <block wx:if="{{hasMore}}">
            <block wx:if="{{loadMoreLoading}}">
              <view class="load-content">
                <text  class="weui-loading"/><text  class="loading-text">加载
中...</text>
              </view>
```

非卖品，仅供非商业用途或交流学习使用

```
            </block>
            <block wx:else>
              <view class="btn-load-more" bindtap="loadMore">
                <text>单击加载更多</text>
              </view>
            </block>
          </block>
          <block wx:else>
            <view class="load-content">
              <text>我是有底线的</text>
            </view>
          </block>
        </view>
      </scroll-view>
```

页面在 scroll-view 组件底部添加了“加载更多”模块，通过 loadMoreLoading 来显示“单击加载更多”按钮，显示“我是有底线的”代表到了最后一页。

comingSoon.wxss 文件中 CSS 样式代码如下所示：

```
/* pages/comingSoon/comingSoon.wxss */
.page-loading{
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%,-50%);
  font-size: 14px;
  color: #666;
}
.page-loading .loading-text{
  display: inline-block;
  vertical-align: middle;
}

/*"css3 svg 背景图 data:image/svg+xml;base64",
图片的内容经过 base64 编码了，data:image/svg+xml;base64 其实是图片的内容。主要目的是减少浏览器和服务器之间的连接数，提高服务器的并发能力！ */
.weui-loading {
  width: 20px;
  height: 20px;
  display: inline-block;
  vertical-align: middle;
  -webkit-animation: weuiLoading 1s steps(12, end) infinite;
  animation: weuiLoading 1s steps(12, end) infinite;
  background: transparent url(
Igd2lkdGg9JzEyMHB4JyBoZWlnaHQ9JzEyMHB4JyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMj
AwMC9zdmciIHZpZXdCb3g9IjAgMCAxMDAgMTAwIj4KICAgIDxyZWN0IHg9IjAiIHk9IjAiIHdpZH
RoPSIxMDAiIGhlaWdodD0iMTAwIiBmaWxsPSJub25lIiBjbGFzcz0iYmsiPjwvcmVjdD4KICAgID
xyZWN0IHg9JzQ2LjUnIHk9JzQwJyB3aWR0aD0nNycgaGVpZ2h0PScyMCcgcng9JzUnIHJ5PSc1Jy
BmaWxsPScjRTlFOUU5JwogICAgICAgdHJhbnNmb3JtPSdyb3RhdGUoMCA1MCA1MCkgdHJhbn
NsYXRlKDAgLTMwKSc+CiAgICA8L3JlY3Q+CiAgICA8cmVjdCB4PSIwIiB5PSIwIiB3aWR0aD0iNy
g9JzcnIGhlaWdodD0nMjAnIHJ4PSc1JyByeT0nNScgZmlsbD0nIzk4OTY5NycKICAgICAgICAgIH
RyYW5zZm9ybT0ncm90YXRlKDMwIDUwIDUwKSB0cmFuc2xhdGUoMCAtMzApJz4KICAgICAgICAgIC
AgICAgICByZXBlYXRDb3VudD0naW5kZWZpbml0ZScvPgogICAgPC9yZWN0PgogICAgPHJlY3QeD
0nNDYuNScgeT0nNDAnIHdpZHRoPSc3JyBoZWlnaHQ9JzIwJyByeD0nNScgcnk9JzUnIGZpbGw9Jy
M5Qjk5OUEnCiAgICAgICAgICB0cmFuc2Zvcm09J3JvdGF0ZSg2MCA1MCA1MCkgdHJhbnNYRlKD
AgLTMwKSc+CiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgID
```

非卖品，仅供非商业用途或交流学习使用

```
wvcmVjdD4KICAgIDxyZWN0IHg9JzQ2LjUnIHk9JzQwJyB3aWR0aD0nNycgaGVpZ2h0PScyMCcgcn
g9JzUnIHJ5PSc1JyBmaWxsPScjQTNBMUEyJwogICAgICAgICAgdHJhbnNmb3JtPSdyb3RhdGUoOT
AgNTAgNTApIHRyYW5zbGF0ZSgwIC0zMCknPgogICAgPC9yZWN0PgogICAgPHJlY3QgeD0nNDYuNS
cgeT0nNDAnIHdpZHRoPSc3JyBoZWlnaHQ9JzIwJyByeD0nNScgcnk9JzUnIGZpbGw9JyNBQkE5QU
EnCiAgICAgICAgICB0cmFuc2Zvcm09J3JvdGF0ZSgxMjAgNTAgNTApIHRyYW5zbGF0ZSgwIC0zMC
knPgogICAgPHJlY3QgeD0nNDYuNScgeT0nNDAnIHdpZHRoPSc3JyBoZWlnaHQ9JzIwJyByeD0nNS
cgeT0nNDAnIHdpZHRoPSc3JyBoZWlnaHQ9JzIwJyByeD0nNScgcnk9JzUnIGZpbGw9JyNCQUI4Qjkn
CiAgICAgICAgICB0cmFuc2Zvcm09J3JvdGF0ZSgxODAgNTAgNTApIHRyYW5zbGF0ZSgwIC0zMC
knPgogICAgPHJlY3QgeD0nNDYuNScgeT0nNDAnIHdpZHRoPSc3JyBoZWlnaHQ9JzIwJyByeD0nNS
c3JyBoZWlnaHQ9JzIwJyByeD0nNScgcnk9JzUnIGZpbGw9JyNDMkMwQzEnCiAgICAgICAgICB0cm
Fuc2Zvcm09J3JvdGF0ZSgyMTAgNTAgNTApIHRyYW5zbGF0ZSgwIC0zMCknPgogICAgPC9yZWN0Pgog
ICAgPHJlY3QgeD0nNDYuNScgeT0nNDAnIHdpZHRoPSc3JyBoZWlnaHQ9JzIwJyByeD0nNScgcnk9JzUnIGZpbGw9JyNDQkNCQ0InCiAgICAgICAgICB0cmFuc2Zvcm09J3JvdGF0ZSgyNDAgNTAgNTApIHRyYW5zbGF0ZSgwIC0zMCknPgogICAgPHJlY3QgeD0nNDYuNScgeT0nNDAnIHdpZHRoPSc3JyBoZWlnaHQ9JzIwJyByeD0nNScgcnk9JzUnIGZpbGw9JyNENEMkQyRDInCiAgICAgICAgICB0cmFuc2Zvcm09J3JvdGF0ZSgyNzAgNTAgNTApIHRyYW5zbGF0ZSgwIC0zMCknPgogICAgPHJlY3QgeD0nNDYuNScgeT0nNDAnIHdpZHRoPSc3JyBoZWlnaHQ9JzIwJyByeD0nNScgcnk9JzUnIGZpbGw9JyNEQURBREEnCiAgICAgICAgICB0cmFuc2Zvcm09J3JvdGF0ZSgzMDAgNTAgNTApIHRyYW5zbGF0ZSgwIC0zMCknPgogICAgPHJlY3QgeD0nNDYuNScgeT0nNDAnIHdpZHRoPSc3JyBoZWlnaHQ9JzIwJyByeD0nNScgcnk9JzUnIGZpbGw9JyNFMkUyRTInCiAgICAgICAgICB0cmFuc2Zvcm09J3JvdGF0ZSgzMzAgNTAgNTApIHRyYW5zbGF0ZSgwIC0zMCknPgogICAgPC9yZWN0Pgo8L3N2Zz4=) no-repeat;
      background-size: 100%;
    }
```

comingSoon.js 文件完整代码如下所示：

```
// pages/comingSoon/comingSoon.js
let apiUrl = "http://wx.leadingdo.com/demo/Film/comingsoon.aspx"

//查询个数
var pageCount = 20
Page({
  data: {

    //json 对象
    films: [],

    //是否有更多
    hasMore: true,

    //loading 显示标记
    showLoading: true,

    //查询开始坐标
    start: 0,

    //是否显示底部加载更多
    loadMoreLoading: false,
  },

  //下拉刷新
  onPullDownRefresh: function () {
    var that = this
    that.setData({
```

非卖品，仅供非商业用途或交流学习使用

```
      films: [],
      showLoading: true,
      start: 0
    })
    that.readFilms()
  },


  //scroll 滚动事件
  scroll: function (e) {
    //console.log(e)
  },

  //页面加载事件
  onLoad: function () {

    //获取电影列表，从 0 开始，20 个
    var that = this
    that.setData({
      showLoading: true,
      start: 0
    })
    that.readFilms()
  },

  //scroll 滚动底部，调用事件(自动加载更多)
  scrolltolower: function () {
    var that = this
    that.loadMore()
  },

  //底部按钮，单击事件，手动加载更多
  loadMore: function () {
    var that = this

    //显示底部加载效果
    that.setData({
      loadMoreLoading: true
    })

    //网络请求，获取内容
    that.readFilms()
  },

  //获取电影列表，从 start 开始，20 个
  readFilms: function () {

    //导航栏显示加载状态
    wx.showNavigationBarLoading();

    // 定义 this 代理，处理网络返回数据，不能直接使用 this
    var that = this;

    //请求网络，获取电影列表
    wx.request({
      url: apiUrl,
      data: {
```

非卖品，仅供非商业用途或交流学习使用

```
          start: this.data.start,
          pageCount: this.data.pageCount
        },
        method: "POST",
        header: {
          //对于 header['content-type'] 为 'application/json' 的数据，会对数据进
          //行 JSON 序列化
          // 对于 header['content-type'] 为 'application/x-www-form-urlencoded'
          //的数据，会将数据转换成 query string
          'content-type': 'application/x-www-form-urlencoded'
        },
        success: function (res) {

          //取消导航栏加载
          wx.hideNavigationBarLoading();

          //获取返回的内容
          var data = res.data
          if (data.subjects.length == 0) {
            that.setData({
              hasMore: false,
            })
          } else {
            that.setData({
              films: that.data.films.concat(data.subjects),
              start: that.data.start + data.subjects.length
            })
          }

          // 取消加载动画
          that.setData({
            showLoading: false
          })

          // 取消底部加载更多效果
          that.setData({
            loadMoreLoading: false
          })
          console.log(that.data.films);
        }
      })
    },

    //查看详情页面
    viewDetail: function (e) {
      var ds = e.currentTarget.dataset;
      wx.navigateTo({
        url: '../detail/detail?id=' + ds.id + '&title=' + ds.title +
'&type=ing'
      })
    },
  })
```

页面加载时，onLoad 方法被调用，在 onLoad 方法内调用 that.readFilms()，用来读取电影列表。ReadFilms 方法通过 wx.request 请求网络，参数有 start（起始坐标）、pageCount（请

非卖品，仅供非商业用途或交流学习使用

求个数，默认 20 个）。method 是 POST 方式。

需要注意 header 的写法，对于 header['content-type'] 为 'application/json' 的数据，会对数据进行 JSON 序列化；对于 header['content-type'] 为 'application/x-www-form-urlencoded' 的数据，会将数据转换成 query string。根据服务器配置，决定使用哪一种。请求成功之后，解析数据，赋值给 films 数组，页面就会刷新显示电影列表。

图 7-12 电影详情页面

7.6.2 电影详情页面的实现

viewDetail 事件是 cell 单元的单击事件，该事件打开 detail 页面，并传值 id、title、type 字段。详情页面显示效果如图 7-12 所示。

关于电影详情页面的实现，完整代码如下所示。

detail.json 文件设置导航栏，代码如下所示：

```
{
    "navigationBarBackgroundColor":"#ffa589",
    "navigationBarTextStyle":"white",
"navigationBarTitleText":"电影详情",
    "backgroundColor":"#eeeeee",
    "backgroundTextStyle":"light"
}
```

detail.wxml 文件设置页面布局，代码如下所示：

```
<!--pages/detail/detail.wxml-->

<!--loading 动画效果，根据 showLoading 来显示-->
<block wx:if="{{showLoading}}">
  <view class="loadingViewClass">
    <text class="imageloadingClass" />
    <text class="loadingText">正在加载中</text>
  </view>
</block>


<!--图片、电影名、评分、导演、主演、类型、国家/地区、看过、想看、剧情简介-->
<block wx:else>
  <view class="container film-detail">
    <view class="film-item">
      <view class="film-image">
        <image src="{{film.images.medium}}"></image>
      </view>
      <view class="film-info">
        <view>
          <text class="film-title">{{film.title}}</text>
          <text class="film-year">{{film.year}}</text>
```

非卖品，仅供非商业用途或交流学习使用

```
        </view>
        <view class="film-rating">
          <block wx:if="{{film.rating.average > 0}}">
            <text class="label">评分: </text>
            <text class="rating">{{film.rating.average}}</text>
          </block>
          <block wx:else>
            <text class="label">暂无评分</text>
          </block>
        </view>
        <view class="directors">
          <text class="label">导演: </text>
          <block wx:for-items="{{film.directors}}" wx:for-item="director">
            <text class="person">{{director.name}}</text>
          </block>
        </view>
        <view class="casts">
          <text class="label">主演: </text>
          <block wx:for-items="{{film.casts}}" wx:for-item="cast">
            <text class="person">{{cast.name}}</text>
          </block>
        </view>
        <view class="genres">
          <text class="label">类型: </text>
          <block wx:for-items="{{film.genres}}" wx:for-item="genre">
            <text class="person">{{genre}}</text>
          </block>
        </view>
        <view class="genres">
          <text class="label">国家/地区: </text>
          <block wx:for-items="{{film.countries}}" wx:for-item="country">
            <text class="person">{{country}}</text>
          </block>
        </view>
        <view class="collect-wish">
          <view>
            看过(
            <text>{{film.collect_count}}</text>)
          </view>
          <view>
            想看(
            <text>{{film.wish_count}}</text>)
          </view>
        </view>
      </view>
    </view>

<!--剧情简介-->
    <view class="summary">
      <text class="title">剧情简介</text>
      <view class="content">
        {{film.summary}}
      </view>


    </view>
  </view>
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
</block>
```

detail.wxss 文件中 CSS 样式代码如下所示：

```
/* pages/detail/detail.wxss */
.film-detail .film-info{
    line-height: 1.2;
}
.film-detail .film-info view{
  margin-bottom: 6px;
}
.summary{
    padding: 0 10px;
}
.summary .title{
    font-size: 16px;
    margin: 15px 0;
}
.summary .content{
    text-indent: 2em;
}
.collect-wish view{
    display: inline-block;
}
.collect-wish view:first-child{
    margin-right: 5px;
}
.collect-wish text{
    color: #666;
}
```

detail.js 脚本文件代码如下所示：

```
// pages/detail/detail.js

let apiUrl = "http://wx.leadingdo.com/demo/Film/detail.aspx"
Page({
  data: {
    film: {},
    showLoading: true,
    options: null
  },
  onLoad: function (options) {
    var that = this
    wx.setNavigationBarTitle({
      title: options.title
    })

    //网络请求
    wx.request({

      url: apiUrl,
      data: {
        fid: options.id,
        pageCount: this.data.pageCount
      },
      method: "POST",
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        header: {
          //对于 header['content-type'] 为 'application/json' 的数据，会对数据进
          //行 JSON 序列化
          //对于 header['content-type'] 为 'application/x-www-form-urlencoded'
          //的数据，会将数据转换成 query string
          'content-type': 'application/x-www-form-urlencoded'
        },
        success: function (res) {
          var data = res.data
          that.setData({
            film: data,
            showLoading: false
          })
        }
      })
    }
  })
```

7.6.3 搜索页面的实现

搜索页面主要实现按照默认、类型进行搜索，单击可以切换。默认是指：电影、演员、导演的名字关键词搜索。类型是指：电影类型，例如爱情、喜剧等关键词进行搜索。

另外，页面显示了热门搜索和热门类型，使用标签布局。

search.json 文件设置导航栏，代码如下所示：

```
{
    "navigationBarBackgroundColor":"#ffa589",
    "navigationBarTextStyle":"white",
    "navigationBarTitleText":"搜索",
    "backgroundColor":"#eeeeee",
    "backgroundTextStyle":"light"
}
```

search.wxml 文件设置页面布局，完整代码如下所示：

```
<!--pages/search/search.wxml-->

<!--搜索框-->
<view class="search-hd">
  <view class="search-area">
    <form bindsubmit="search">
      <view class="search-type" bindtap="changeSearchType">{{searchType ==
0 ? '默认▼' : '类型▼'}}</view>
      <input class="search-txt" name="keyword" auto-focus placeholder=
"{{searchType == 0 ? '电影、演员或导演' : '影片类型，如：爱情、喜剧'}}" />
      <button class="search-btn" formType="submit">搜索</button>
    </form>
  </view>

  <!--热门搜索-->
  <view class="search-keyword">
    <view class="search-keyword-title">热门搜索</view>
    <view wx:for="{{hotKeyword}}" wx:for-item="hotKeywordItem" wx:key=
```

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
"hotKeywordItem" class="search-keyword-item" data-keyword="{{hotKeywordItem}}"
bindtap="searchByKeyword">{{hotKeywordItem}}</view>

        <!--热门类型-->
        <view class="search-keyword-title">热门类型</view>
        <view wx:for="{{hotTag}}" wx:for-item="hotTagItem" wx:key="hotTagItem"
class="search-keyword-item" data-keyword="{{hotTagItem}}" bindtap="searchByTag">
{{hotTagItem}}</view>
      </view>
    </view>
```

search.wxss 文件中 CSS 样式完整代码如下所示：

```
/* pages/search/search.wxss */

.search-hd {
    width: 100%;
    height: 140rpx;
    background-color: #fff;
    border-bottom: 1px solid #ffa589;
}
.search-area {
    box-sizing: border-box;
    position: relative;
    margin: 30rpx;
}

.search-type {
    position: absolute;
    z-index: 11;
    top: 0;
    left: 0;
    width: 120rpx;
    height: 80rpx;
    line-height: 80rpx;
    text-align: center;
    border-right: 1px solid #ffa589;
}

.search-txt {
    box-sizing: border-box;
    width: 100%;
    height: 80rpx;
    padding-left: 140rpx;
    padding-right: 20rpx;
    border-radius: 10rpx;
    border: 1px solid #ffa589;
    font-family: "微软雅黑";
    color: #ffa589;
}

.search-btn {
    position: absolute;
    z-index: 11;
    top: 0;
    right: 0;
    height: 80rpx;
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    width: 120rpx;
    text-align: center;
    line-height: 80rpx;
    font-size: 28rpx;
    border-bottom-left-radius: 0;
    border-top-left-radius: 0;
    color: #ffa589;
    border: 1px solid #ffa589;
}

.search-btn::after{
    border-bottom-left-radius: 0;
    border-top-left-radius: 0;
}

.search-keyword {
    padding: 30rpx;
}

.search-keyword-title {
    padding-left: 20rpx;
    margin-top: 40rpx;
    margin-bottom: 20rpx;
    font-size: 30rpx;
    border-left: 10rpx solid #ffa589;
}

.search-keyword-item {
    display: inline-block;
    padding: 10rpx;
    border: 1px solid #ffa589;
    border-radius: 15rpx;
    background-color: #fbfbfb;
    margin: 0 10rpx 10rpx 0;
    color: #ffa589;
}

.message-area {
    position: fixed;
    width: 100%;
    height: 100%;
    z-index: 99;
}
.message {
    box-sizing: border-box;
    position: fixed;
    z-index: 999;
    left: 50%;
    top: 50%;
    width: 200rpx;
    height: 200rpx;
    padding: 30rpx;
    margin-top: -100rpx;
    margin-left: -100rpx;
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    border-radius: 16rpx;
    background-color: rgba(0, 0, 0, .75);
    color: #ffa589;
    animation: fadeIn .3s;
}

.message-icon {
    height: 60rpx;
    width: 60rpx;
    background-position: center;
    background-repeat: no-repeat;
    background-size: 100rpx;
}

.message-icon-success {
    background-image: '../../image/weui-success-icon.png';
}

.message-icon-warning {
    background-image: '../../image/weui-warning-icon.png';
}

.message-icon-info {
    background-image: '../../image/weui-info-icon.png';
}

.message-content {
    margin-top: 15rpx;
    text-align: center;
}
```

search.js 脚本文件完整代码如下所示：

```
// pages/search/search.js

Page({
  data: {
    searchType: 0,
    hotKeyword: ['看不见的客人', '血战钢锯岭', '海边的曼彻斯特', '你的名字', '爱乐之城', '极品基老伴：完结篇', '比利·林恩的中场战事', '比海更深', '驴得水', '杰出公民', '斯隆女士','隐藏人物','釜山行','萨利机长','欢乐好声音','伦敦一家人','湄公河行动','神奇动物在哪里','怒','西葫芦的生活'],
    hotTag: ['爱情', '悬疑','动作', '喜剧','科幻','恐怖' ],

    //提醒使用
    showTipTxt: '',
    tipHidden: true
  },


  //选择搜索类型
  //默认：电影、演员或导演名称
  //类型：影片类型，如：爱情、喜剧
  changeSearchType: function () {
    var types = ['默认', '类型'];
    var that = this
    wx.showActionSheet({
      itemList: types,
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
        success: function (res) {
          if (!res.cancel) {
            that.setData({
              searchType: res.tapIndex
            })
          }
        }
      })
    },

    //搜索
    search: function (e) {
      var that = this
      var keyword = e.detail.value.keyword
      if (keyword == '') {

        wx.showToast({
          title: '请输入搜索内容',
          icon: 'success_no_circle',
          duration: 2000
        })

        return false
      } else {
      that.tosearchResult(this.data.searchType,keyword)
      }
    },


    // 热门搜索 单击事件
    searchByKeyword: function (e) {
      var that = this
      var keyword = e.currentTarget.dataset.keyword
      that.tosearchResult("0",keyword)
    },


    // 热门标签 单击事件
    searchByTag: function (e) {
      var that = this
      var keyword = e.currentTarget.dataset.keyword
      that.tosearchResult("1",keyword)
    },

    //界面跳转
    tosearchResult: function (sType,keyword) {
      wx.navigateTo({
        url: '../searchResult/searchResult?searchType=' + sType + '&keyword=' +
keyword
      })
    }
  })
```

关键词搜索、热门搜索、热门类型，单击之后都会调用 tosearchResult 事件，tosearchResult 事件打开 searchResult 页面，传值 searchType、keyword。searchType=0 表示按照默认搜索，searchType=1 表示按照电影类型搜索。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！

searchResult 页面主要是按照参数调用接口，获取相关的电影类别，页面显示和“正在热映”“即将上映”的页面显示一样。这里主要看一下 searchResult.js 脚本文件代码，代码如下所示：

```
// pages/searchResult/searchResult.js

let apiUrl = "http://wx.leadingdo.com/demo/Film/search.aspx"

Page({
data: {
//json 对象
films: [],

//是否有更多
hasMore: true,

//loading 显示标记
showLoading: true,

// 搜索类型
searchType: '0',

// 关键词
keyword: ''
},


//页面加载事件
onLoad: function (options) {
var that = this
that.setData({
searchType: options.searchType,
keyword: options.keyword,
title: options.keyword
})

console.log('=========', this.data.searchType, this.data.keyword, );

//获取电影列表，从 0 开始，20 个
var that = this
that.setData({
showLoading: true
})
that.readFilms()
},

//下拉刷新
onPullDownRefresh: function () {
var that = this
that.setData({
films: [],
showLoading: true,
})

that.readFilms()
},
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
//scroll 滚动事件
scroll: function (e) {
//console.log(e)
},

//获取电影列表，从 start 开始，20 个
readFilms: function () {

//导航栏显示加载状态
wx.showNavigationBarLoading();

// 定义 this 代理，处理网络返回数据，不能直接使用 this
var that = this;

//请求网络，获取电影列表
wx.request({
url: apiUrl,
data: {
searchType: this.data.searchType,
keyword: this.data.keyword,
},
method: "POST",
header: {
//对于 header['content-type'] 为 'application/json' 的数据，会对数据进行 JSON 序列化
//对于 header['content-type'] 为 'application/x-www-form-urlencoded' 的数
//据，会将数据转换成 query string
'content-type': 'application/x-www-form-urlencoded'
},
success: function (res) {

//取消导航栏加载
wx.hideNavigationBarLoading();

//获取返回的内容
var data = res.data
if (data.subjects.length == 0) {

} else {
that.setData({
films: that.data.films.concat(data.subjects),
start: that.data.start + data.subjects.length
})
}

// 取消加载动画
that.setData({
showLoading: false
})

console.log(that.data.films);
}
})
},

//查看详情页面
viewDetail: function (e) {
```

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！


```
    var ds = e.currentTarget.dataset;
    wx.navigateTo({
    url: '../detail/detail?id=' + ds.id + '&title=' + ds.title + '&type=ing'
    })
    }
    })
```

7.7 总结

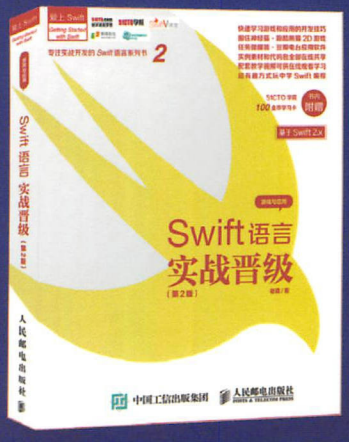
到这里，本书已将微信小程序开发中常用的一些功能介绍完毕。相信你也通过本书将微信小程序开发中常用的理论知识都学会了，并将自己动手实现一些功能。恭喜你，你已成功地加入到微信小程序开发队伍中了。

微信小程序对于初学者也很容易上手。但学习微信小程序没有捷径，关键在于多练习多做，不怕走弯路，就怕不肯动手。对于微信小程序来说，开发软件和语法的更新速度很快，这也需要大家以后继续学习，不断充电。

微信小程序作为一门编程语言，总在不断地升级，在每一次迭代中，很多语法也有所调整。大家在查看源代码的时候，可能会出现一些错误，希望大家不要纠结这些错误，项目报错，是由于微信在升级小程序语言的时候，重新定义了一些语法的使用规范。大家可以在 QQ 群里面一起探讨，或者在网上查询资料来解决这些问题。

非卖品，仅供非商业用途或交流学习使用

版权所有，严禁以任何方式传播本PDF，违者自负法律责任！



从零开始快速掌握微信小程序项目的开发方法

完整的实例解析带你完成从理论到实践的蜕变

一本书玩转微信小程序开发

- 容易上手，通过案例精细讲解小程序语言的实战技巧，使读者容易理解，并能马上学以致用。对于每一部分具体内容，都精心设计了相应的示例程序，一方面可以帮助读者加深理解，另一方面也可以逐步培养读者的程序设计能力。
- 内容全面，本书详细讲解了开发工具的使用、基本组件、API 的使用。
- 技术实用，通过多个案例详细讲解了小程序的开发过程和代码实现，从 0 到 1 开发属于自己的小应用。

封面设计：郭　媛
分类建议：计算机 / 程序开发
人民邮电出版社网址 :www.ptpress.com.cn

ISBN 978-7-115-49328-6
定价：89.00 元